JPL D-7669, Part 2

Planetary Data System Standards Reference

June 1, 1999
Version 3.3

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

TABLE OF CONTENTS

PDS Standards Reference Change Log

Version	Section	Change
3.1	1.1	PDS Data Policy added
	2.3	Reference coordinate standard expanded to support body-fixed rotating, body-fixed non-rotating, and inertial coordinate systems.
	2.4	Ring coordinate standard added.
	3.0	List of internal representations of data types moved to Appendix C
	3.2	EBCDIC_CHARACTER added to PDS Standard data types
	5.2.3	Minimal label option described
	6.3	Data set collection naming -- data processing level component made optional
	6.4	Data set naming -- added support for SPICE and Engineering, where no instrument component applies
	10.0, ALL	PDS use of UNIX/POSIX forward slash separator for path names. VMS-style bracket notation replaced.
	10.2.1	Required file names for catalog objects included
	12.5.4.2	PDS use of double quotes clarified
	13.2	Use of Primitive objects described
	14	New chapter -- Pointer Usage
	17	New chapter -- PDS Usage of N/A, UNK, and NULL
	19	Logical Volume organization added
	Appendix A	Primitive Objects added
	Appendix A	Header object -- required and optional keyword lists changed Container object -- Column no longer a requried sub-object

	Appendix B	Streamlined Catalog Object Templates with examples replace 3.0 set
	Appendix C	New appendix containing internal representations of data types (moved from Chapter 3)
	Appendix D	Outline and example for AAREADME.TXT added
	Appendix E	Version 3.0 Acronyms and Abbreviations modified and moved to this Appendix. Spelling and Word Usage section deleted.
	Index	The document now features an index.
	ALL	No other substantive changes have been made to the standards since the release of Version 3.0. Throughout the document, clarifications have been made, typos corrected, some sections have been rearranged, and new examples have been supplied.

Version	**Section**	**Change**
3.2		Release Date: 7/24/95
	5.1.2	Label format discussion added Noted that values in labels should be upper case (except descriptions). Fixed examples in Appendix A.
	5.2.3, Appendix A	Noted that for data products using minimal labels, DATA_OBJECT_TYPE = FILE in the Data Set Catalog Template
	6	Added target IDs for DUST and SKY Added instrument component values SEDR and POS Noted that Data Set and Data Set Collection IDs and Names should be upper case. Fixed examples.
	8 and 19	Listed CALIB and GEOMETRY as recommended directory names (as opposed to required).
	8.2	SOFTWARE Subdirectory naming recommendation added
	9.1	Volumes may contain multiple versions of VOLINFO

9.2.1	Increased maximum line length in text file to 78 characters plus CR/LF
10.1	Clarified file name spcification. Noted that file name must be upper case and that full stop character required
10.2	Added recommendation that file extension identify the data type of a file. Added .QUB as reserved file extension for spectral image qubes. Added SPICE file extensions to reserved file extension list. catalog pointer name and file name: SWINV.CAT Added LABINFO.TXT to list of required xxxINFO.TXT files. Added recommended xxx INFO.TXT file names for SOFTWARE subdirectories.
10.2.3 and 5.1	added note that detached label file (*.LBL) should have the same base name as the associated data file
11.1.1	Added PDS Extended Attribute Record (XAR) policy
11.1.2	Added recommendation that CDs be premastered using single-session, single-track format.
11.1.3	Added section on Packaging Software files on a CD-ROM
14.1.2	Added new example of structure pointer
15	Added recommendation that for VAX/VMS-compatible CDs, fixed length and variable length files be an even number of bytes. Removed reference to VMS restriction to an even number of bytes in section 15.2
15.1	Removed discussion of use of BLOCK_BYTES and BLOCKING_TYPE (since this data element not in PSDD)
15.3	Added notation that CR/LF is required line terminator for PDS label and catalog files
15.5	Reworded first sentence.
17.2	Allow definition of numeric constants representing N/A, UNK, and NULL to be defined for use in an INDEX table.
18	replaced reference to PDS V1.0 with a general statement

19	Added SOFTWARE subdirectory recommendations
19	Recommend that an archive volume be based on a single version of the PDS standards. Volume organization guidelines added.
19.2	Clarified requirements for files & directories when logical volumes used
19.3	INDEX table standard update
19.3	use of axx- and bxx- prefixes in required file names clarified
19.4, Appendix A	fixed examples–Volume and Volume set names capitalized
19.5.1	Volume set ID formation rule modified.
Appendix A	updated COLUMN, BIT_COLUMN, and HISTOGRAM objects required and optional keyword lists to be consistent with Table 3.1
Appendix A	Added ALIAS and INDEX_TABLE objects
Appendix A	Added examples of COLUMN objects having ITEMs
Appendix A	Clarified use of ROW_SUFFIX_BYTES and ROW_PREFIX_BYTES for SPARE fields in Tables with fixed length records
Appendix A	Clarified the requirements for VOLUME objects for Logical volumes
Appendix A	Fixed examples using HEADER object to conform to current standard. Modified description of Header object to eliminate confusion..
Appendix B	Inventory, Software_Inventory and Target templates added
Appendix B	Removed incorrect example of use of Personnel template
Appendix D	INDXINFO.TXT and SOFTINFO.TXT outlines and examples added
Appendix D.1	Modified example of AAREADME.TXT to include rules on how pointer statements are resolved.

	Appendix E and F	Added Appendix E - NAIF Toolkit Directory Structure. Acronyms and Abbreviations moved to Appendix F.
	ALL	corrected typos, clarified text, added rationale for some standards, updated examples to conform to latest standards
	Change Log	Version 3.1 change log updated--some items were missing

Version	**Section**	**Change**
3.3		Release Date: 6/1/99
	1.0	Added DVD as new medium
	1.3	Changed Version to 3.3
	1.6	Updated/corrected references
	1.7	Added reference to PDS web page
	2.0	Added definition for IAU Clarified text
	2.3	Corrected punctuation
	2.7	Fixed punctuation for references
	3.4	Corrected punctuation
	3.7	Corrected spelling and punctuation
	4.0	Added Section headers for Primary & Secondary Objects
	4.1	Corrected paragraph formatting
	5.1.2	Added paragraph about ASCII character set Added paragraph about Label Padding Fixed math in calculating start byte of 8th record Aligned keyword/values
	5.2.2	Corrected grammar
	5.2.3	Removed "’"in the Data Set catalog template.
	5.3.1	Changed Version to 3.3
	5.3.2	Modified last paragraph
	5.3.3	Listed examples of primary and secondary objects
	5.3.3.2	Changed ’bottom’ to ’following’
	5.3.4	Removed AMMOS as an example
	5.3.4.1	Removed SPACECRAFT_NAME as valid keyword
	5.3.4.3	Removed SPACECRAFT_NAME as valid keyword.
	5.3.5	Changed PDS has developed and continues to develop... Added example for a pointer (^DESCRIPTION)
	5.3.6	Aligned keyword/values Clarified statement
	5.3.7	Changed: needed for conformance
	6.0	Prioritized organizations that PDS works with
	6.1	Provided definition for Data Set Collection and removed MGN example.

	Corrected spelling (considerations) and punctuation
6.2	Added acronyms for data set name and identifier
6.3	Changed paragraph from future tense to past tense
6.4	Section 5 - comets
	Section 6 - added acronyms to list
	Section 6 - corrected spelling (ephemeris)
	Section 7 - corrected spelling (gravity)
	Section 8 - clarified version number rules
7.0	Updated paragraph
7.1	Clarified statements about date/time formats
7.2.1	Added PDS preference for convention
7.3.1	Corrected grammar
	Reformatted paragraph
7.3.2	Corrected grammar
	Updated paragraphs
8.1	Corrected grammar (standards directory)
	Added EXTRAS directory
	Added Browse and Data directory descriptions
8.2	Section 4 - Better examples of directory names
	Section 5 - Reformatted paragraph
	Section 8 - Corrected spelling and grammar
8.3	Changed to valid keywords
8.4	Corrected grammar (data are)
9.0 - 9.3.3	Complete rewrite of Documentation Standard
	Added HTML standards
10.0 - 10.1	Added ISO 9660 Level 2 description
	Added ";1" to Level 1 description
10.2.1	Clarified required file names paragraphs
	Added TARGET_CATALOG pointer to list
10.2.2	VOLDESC.SFD file becomes deprecated
10.2.3	Described detached label
	Corrected grammar (its)
10.2.4	Added extensions and changed SPICE extensions
	Corrected spelling (postscript) and grammar (data that have)
11.1.1	Changed chapter name
12.1	Aligned equal signs
12.1.1.1	Added reference
12.2	Reformatted paragraph
12.3	Spelling
12.3.1	Corrected punctuation (1.234E2)
12.3.1.2	Corrected value (16#+4B#)
	Reformatted paragraph
12.3.1.3	Corrected value (1.234E3)
12.3.2	Updated paragraphs
12.3.2.1	Clarified date format
12.3.2.3	Clarified paragraph

12.3.2.4	Changed year to 4 digits
12.3.2.5	Updated paragraph
12.3.2.5.1	Corrected value (1990-158T15:24:12z)
12.3.3.1	Corrected value ("::=")
12.3.4	Added examples
12.3.5	Corrected punctuation and grammar (units)
12.4	Corrected punctuation
12.4.1	Corrected grammar (the the)
	Aligned equal signs
12.4.2	Aligned equal signs
12.5.2	Reformatted asterisks to not be superscript
	Corrected value (60.15)
12.5.3.1	Corrected grammar (affect)
	Reformatted paragraphs
12.5.4	Corrected value (IO)
	Added valid quoted strings
12.5.4.1	Clarified paragraph
12.5.5	Reformatted asterisk to not be superscript
	Corrected spelling (eccentricity)
	Changed to valid keyword
12.5.6	Corrected value (removed 1st bracket "[")
	Changed to valid keyword
12.6	Reformatted paragraphs
12.7	Reformatted paragraphs
12.7.1	Corrected grammar (sections detail)
12.7.2	Corrected grammar ("is that are")
13.1	Added required keywords to definition
14.1.1	Corrected grammar (occurs)
14.1.2	Corrected punctuation
	Corrected value (^STRUCTURE)
	Changed paragraph numbering
14.2	Reformatted pointer rules
15.0	Reformatted paragraph and table
15.2	Changed paragraph numbering
15.3	Changed paragraph numbering
16.0	Corrected grammar
16.2	Clarified paragraph
	Changed case of #mark#
17.1	Changed case of title (and)
17.1.2	Corrected punctuation (information)
17.2	Corrected case of title (and)
18.0	Corrected SI Units (electricity potential, etc)
	Updated paragraph
19.1	Corrected grammar (volume types)
	Corrected grammar (up to the)
19.3	Corrected grammar (an SFDU)

	Corrected spelling (global)
	Updated Catalog and Index definitions
	Added description of the EXTRAS directory
	Added Preferred Method for supplying PDS catalog objects
19.4.1	Corrected grammar (data have been)
	Changed case of value (ID)
19.5	Corrected spelling (radiometry)
	Corrected value (VOLUME_SET_NAME)
	Corrected value (VOLUME_SET_ID)
19.5.1	Reformatted paragraph
19.7	Corrected case of value (IDs)
20.0 - 20.6	Complete rewrite of Zip Compression
Appendix A	Added URL to Cold Fusion pages
A.1	Updated definition for ALIAS
	Corrected spelling (subobject)
A.2	Added and changed Optional keywords
	Reformatted paragraphs
	Corrected spelling (the time)
A.3	Changed Optional keywords
	Corrected spelling (created)
A.5	Added TARGET to Optional Objects
	Clarified use of CATALOG.CAT
	Formatted paragraph
A.7	Formatted paragraph
	Changed Optional keywords
A.8	Updated paragraph
A.10	Changed case of keyword values to uppercase
A.11	Corrected grammar (on a)
	Corrected grammar (on the medium)
A.12	Removed incorrect statements
	Updated example
A.13	Changed Optional keywords
A.14	Removed a Required keyword
	Added Optional keywords
A.15	Changed value to keyword (GAZETTEER_TABLE)
	Corrected grammar (the breath & upper right)
	Added Optional Keywords section
	Added Optional Objects section
	Added trailing double quote to DESCRIPTION section
A.16	Corrected paragraph to reflect proper file name
	Changed value to be enclosed in double quotes
A.18	Added Required and Optional Keywords and Objects sections
A.19	Added BAND_NAME keyword
	Added Optional keyword
	Changed values to be keyword (CHECKSUM)
	Changed values to be keyword (SCALING_FACTOR)

A.20	Changed paragraphs
	Changed case of keyword values to uppercase
A.21	Reformatted paragraphs
	Removed Optional Keyword
	Added Optional Objects
	Corrected example (see additional example in A.27.1)
A.23	Added example for CORE_ITEM_TYPE
	Corrected FILE_RECORDS to be accurate
	Corrected invalid keyword (SUB_SOLAR_AZIM UTH)
A.24	Corrected grammar (data that vary)
A.26	Corrected grammar (data are)
	Corrected punctuation (The Tookit)
A.27	Corrected grammar (meta-data which are)
	Updated section numbers to reflect location (spares)
	Repaired examples (byte lengths)
A.28	Line length to 72 chars
	Added Required and Optional Objects
	Repaired example
A.29	Updated Optional keyword
	Changed case of keyword values to uppercase
Appendix B	Changed paragraph
	Changed text description length to be 80 characters from 72
	Added text formatting standards
B.1	Corrected punctuation
	Repaired example
B.2	Reformatted paragraph
	Reformatted and repaired example
B.3	Corrected spelling (DESCRIPTION)
	Reformatted paragraph
	Reformatted and repaired example
B.4	Corrected spelling (description & instrument)
	Reformatted paragraph
	Reformatted and repaired example
B.5	Corrected grammar (properties of the)
	Reformatted paragraph
	Reformatted and repaired example
B.6	Repaired example
B.7	Reformatted paragraph
	Reformatted and repaired example
B.8	Repaired example
B.10	Corrected spelling (package)
	Replaced example of SOFTWARE_INVENTORY template
B.11	Corrected grammar (target catalog)
	Corrected grammar (SURFACE_GRAVITY)
	Repaired example
Appendix C	Minor corrections throughout text

C.5	Corrected spelling (exponent-as-stored)
C.10	Corrected spelling (imaginary)
Appendix E	Corrected sentence (source code for) Corrected spelling (spacit) Corrected grammar (These data are) Corrected punctuation
Appendix F	Corrected CD-WO nomenclature Added DE (Data Engineer) Corrected spelling (Principal)
Appendix G	Added SAVED Data as new section

Chapter 1

Introduction

In order for planetary science data to be used by those not involved with its creation, certain supporting information must be available with the data. Such information enables effective data access and interpretation. Therefore, standards regarding the quality and completeness of data must be enforced. Also, the interchange of data is increasingly important in planetary science. Electronic communication mechanisms have grown in sophistication, and the use of new media (such as CD-ROMs and DVD) for data storage and transfer requires format standards to ensure readability and usability. The Planetary Data System (PDS) has therefore developed a nomenclature that is consistent across discipline boundaries, as well as standards for labeling data files.

1.1 PDS Data Policy

Only data that complies with PDS standards will be published in volumes which are labelled "Conforms to PDS Standards". Non-compliant data published in recognized formats should be authored by the publishing institution with PDS participation acknowledged only as "funded by PDS". The PDS Management Council will make decisions on compliance waivers. Non-compliant data sets will be permitted only under unusual circumstances.

1.2 Purpose

This document is intended as a reference manual to be used in conjunction with the *PDS Data Preparation Workbook* and the *Planetary Science Data Dictionary*. The *PDS Data Preparation Workbook* describes the end-to-end process for submitting data to the PDS and gives instructions for preparing data sets. In addition, a glossary of terms used throughout this document is contained as an appendix to the workbook. The *Planetary Science Data Dictionary* contains definitions of the standard data element names and objects. This reference document defines all PDS standards for data preparation.

1.3 Scope

The information included here constitutes Version 3.3 of the Planetary Data System data preparation standards for producing archive quality data sets.

1.4 Audience

This document is intended primarily to serve the community of scientists and engineers responsible for preparing planetary science data sets for submission to the PDS. These include restored data from the era prior to PDS, mission data from active and future planetary missions, and data from earth-based sites. The audience includes personnel at PDS Discipline and Data Nodes, mission Principal Investigators, and Ground Data System engineers.

1.5 Document Organization

The first chapter of this document, Chapter 1 - Introduction, provides introductory material and lists of other reference documents. The remaining chapters provide a dictionary of data preparation standards, organized alphabetically by standard name.

1.6 Other Reference Documents

The following reference sources are mentioned in this document:

- Batson, R. M., (1987) "Digital Cartography of the Planets: its Status and Future", *Photogrammetric Engineering & Remote Sensing 53,* 1211-1218.
- Davies, M.E., *et al* (1991) "Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1991", *Celestial Mechanics*, 53,377-397.
- Greeley, R. and Batson, R.M. (1990) *Planetary Mapping*, Cambridge University Press, Cambridge, 296p.
- *Guide on Data Entity Naming Conventions,* NBS Special Publication 500-149.
- *Planetary Science Data Dictionary,* JPL D-7116 Rev D, July 15, 1996, (Available from PDS).
- *Planetary Data System Data Preparation Workbook Version 3.1,* JPL D-7669 Part 1, February 17, 1995, (Available from PDS)
- *Issues and Recommendations Associated with Distributed Computation and Data Management Systems for the Space Sciences,* National Academy Press, Washington, DC, 111p.

International Standards Organization (ISO) References

- ISO 9660:1988 “Information Processing - Volume and File Structure of CD-ROM for Information Exchange”, April 15, 1988.
- ISO 646:1991 ASCII character set.

- ISO 8601:1988 “Data Element and Interchange Formats – Representations of Dates and Times”

SFDU and PVL References

- *Standard Formatted Data Units - Structure and Construction Rules,* CCSDS 620.0-R-1.1c, May 1992.
- *Standard Formatted Data Units - A Tutorial;* CCSDS 620.0-G-1, May 1992.
- *Parameter Value Language Specification (ccsd0006);* CCSD 641.0-R-0.2, June 1991.
- *Parameter Value Language -- A Tutorial;* CCSDS 641.0-G-1.0, May 1992.

1.7 Online Document Availability

The *Planetary Science Data Dictionary, Planetary Data System Data Preparation Workbook*, and this document, the *Planetary Data System Standards Reference* are available online. Information on accessing these references may be found on the PDS website, which is located at:

http://pds.jpl.nasa.gov

To obtain a copy of these documents, or for questions concerning these documents, contact the PDS Operator, or a PDS data engineer.

Chapter 2

Cartographic Standards

The following cartographic data standards were developed through an iterative process involving both the NASA Planetary Cartography Working Group (PCWG) and the PDS. Members of the PCWG are also on the key International Astronomical Union (IAU) committees which set these same standards for international adoption; therefore, the PDS-adopted cartographic standards are consistent with the IAU standards. The PDS, rather than making unilateral decisions on cartographic data standards, looks to the PCWG as the controlling body for these standards within NASA and the PDS. It is recognized that the IAU continually reviews its standards and may, at some time, make a change affecting the cartographic standards. If this happens, the PDS will work with the PCWG and decide its own course of action at that time.

Cartographic standards used in a data set should be identified, and where helpful, documented on an archive volume.

2.1 Inertial Reference Frame/Timetag/Units

The Earth Mean Equator and Equinox of Julian Date 2451545.0 (referred to as the "J2000" system) is the standard inertial reference frame. The Earth Mean Equator and Equinox of Besselian 1950 (JD 2433282.5) is also to be supported because of the wealth of previous mission data referenced to this system. The transformations between the two systems are to be available. Time tagging of data using UTC in Year, Month, Day, Hour, Minute and decimal Seconds is the standard, with Julian Date being supported. SI metric units, including decimal degrees, are the standard.

2.2 Spin Axes and Prime Meridians

The IAU-defined spin axes and prime meridians defined relative to the J2000 Inertial Reference System are the standard for planets, satellites and asteroids where these parameters are defined. For other planetary bodies, definitions of spin axes and prime meridians determined in the future should have the body-fixed axes aligned with the principal moments of inertia,with the North Pole defined as along the spin axis and above the Invariable Plane. Where insufficient observations exist for a body to determine the principal moments of inertia, coordinates of a surface feature will be specified and used to define the prime meridian. It is expected that some small, irregular bodies may have chaotic rotations and will need to be handled on a case-by-case basis.

2.3 Reference Coordinates

There are three basic types of coordinate systems, body-fixed rotating, body-fixed non-rotating and inertial. A body-fixed coordinate system is one associated with the body (e.g. planetary body or satellite). In contrast to inertial coordinate systems, the body-fixed system is centered on the body and rotates with the body (unless it is a non-rotating type), whereas the inertial coordinate system is fixed at some point in space.

To support the descriptions of these reference coordinate systems, the PDS has defined the following set of data elements (See *Planetary Science Data Dictionary* for complete definitions.):

COORDINATE_SYSTEM_TYPE
COORDINATE_SYSTEM_NAME
LATITUDE
LONGITUDE
EASTERNMOST_LONGITUDE
WESTERNMOST_LONGITUDE
MINIMUM_LATITUDE
MAXIMUM_LATITUDE
POSITIVE_LONGITUDE_DIRECTION

Currently, PDS has specifically defined two types of body-fixed rotating coordinate systems, Planetocentric and Planetographic. However, the set of related data elements are modelled such that definitions for other body-fixed rotating coordinate systems, body-fixed non-rotating and inertial coordinate systems can be added when the need arises. If this is the case, contact a PDS data engineer for assistance.

The definition of Planetographic longitude is dependent upon the rotation direction of the body, with longitude being measured as increasing in the direction opposite to the rotation. That is to say that the longitude increases to the west if the rotation is prograde (or eastward) and vice versa. Table 2.1 lists the rotation direction (prograde or retrograde) of the primary planetary bodies and the Earth's moon. It also indicates the valid longitude range for each body. In order to accommodate different traditions in measuring longitude, in the *Planetary Science Data Dictionary*, PDS defines a broad longitude range: (-180, 360). Table 2.1 indicates which part of that range is applicable to which body.

Table 2.1: Primary Bodies and Earth's Moon - Rotation Direction and Longitude Range

Planet	Rotation Direction	Longitude Range
Earth	Prograde	(0, 360) (-180, 180)*
Mars	Prograde	(0, 360)
Mercury	Prograde	(0, 360)
Moon	Prograde	(0, 360) (-180, 180)*
Jupiter	Prograde	(0, 360)
Neptune	Prograde	(0, 360)
Pluto	Retrograde	(0, 360)
Saturn	Prograde	(0, 360)
Sun	Prograde	(0, 360) (-180, 180)*
Uranus	Retrograde	(0, 360)
Venus	Retrograde	(0, 360)

* The rotations of the Earth, Moon and Sun are prograde, however it has been tradition to measure longitudes for these bodies as increasing to the east instead of the west. PDS recommends that the Planetographic longitude standard be followed, but it also will support the tradition. Therefore, the longitude range of (-180, 180) is supported for the Earth, Moon and Sun.

2.3.1 Body-Fixed Rotating Coordinate Systems

2.3.1.1 Planetocentric

The Planetocentric system has an origin at the center of mass of the body. Planetocentric latitude is the angle between the equatorial plane and a vector connecting the point of interest and the origin of the coordinate system. Latitudes are defined to be positive in the northern hemisphere of the body, where north is in the direction of Earth's angular momentum vector, i.e., pointing toward the hemisphere north of the solar system invariant plane. Longitudes increase toward the east, making the Planetocentric system right-handed.

2.3.1.2 Planetographic

The Planetographic system has an origin at the center of mass of the body. The planetographic latitude is the angle between the equatorial plane and a vector through the point of interest, where the vector is normal to a biaxial ellipsoid reference surface. Planetographic longitude is defined to increase with time to an observer fixed in space above the object of interest. Thus, for prograde rotators (rotating counter clockwise as seen from a fixed observer located in the hemisphere to the north of the solar system invariant plane), planetographic longitude increases toward the west. For a retrograde rotator, planetographic longitude increases toward the east.

2.4 Rings

Locations in planetary ring systems are specified in polar coordinates by a radius distance (measured from the center of the planet) and a longitude. Longitudes increase in the direction of orbital motion, so the ring pole points in the direction of right-handed rotation. Note that this corresponds to the IAU-defined north pole for Jupiter, Saturn and Neptune but the south pole for Uranus.

Longitudes are given relative to the ascending node of the ring plane on the Earth's mean equator of J2000. However, the Earth's mean equator of B1950 is also supported as a reference longitude because of the wealth of data already reduced using this coordinate frame. The difference is generally a small, constant offset to the longitude. All longitude values fall between 0 and 360 degrees.

Note that ring coordinates are always given in an inertial frame. It is impossible to define a suitable rotating coordinate frame for a ring system because features rotate at different rates. When it is necessary to specify the location of a moving body or feature, one must give the rotation rate and the epoch in addition to the longitude.

The *Planetary Science Data Dictionary* (PSDD) contains a set of data elements designed to describe ring-related longitudes. Please see the PSDD for these elements and their complete definitions.

2.5 Reference Surface

The Digital Terrain Model (DTM), giving body radius as a function of Cartographic latitude and longitude in a sinusoidal equal-area projection, is the standard. Mars is to be an exception where Planetographic latitude is to be used. Spheroids, ellipsoids and harmonic expansions giving analytic expressions for radius as a function of Cartographic coordinates are to be supported.

The Digital Image Model (DIM) giving body "brightness" in a specified spectral band or bands as a function of Cartographic latitude and longitude in a sinusoidal equal-area projection, and associated with the surface radius values in the DTM, is the standard. Mars is to be an exception where Planetographic latitude is to be used. DIMs registered to spheroids, ellipsoids and harmonic expansions are to be supported.

2.6 Map Resolution

The suggested spatial resolution of a map is $1 / 2^n$ degrees. The suggested vertical resolution is 1×10^m meters, with m and n chosen to preserve all the resolution inherent in the data.

2.7 References

The following references give more detail on the cartographic data standards:

Davis, M. E., et al (1991) "Report of the IAU/IAG/COSPAR Working Group on Cartographic Coordinates and Rotational Elements of the Planets and Satellites: 1991," *Celestial Mechanics*, 53, 377-397.

Batson, R.M., (1987) "Digital Cartography of the Planets: New Methods, its Status and Future", *Photogrammetric Engineering & Remote Sensing*, 53, 1211-1218.

Greeley, R. and Batson, R.M. (1990) *Planetary Mapping*, Cambridge University Press, Cambridge, 296p.

Chapter 3

Data Type Definitions

Each PDS-archived product is described using label objects that provide information about the data types of stored values. The data elements DATA_TYPE, BIT_DATA_TYPE, and SAMPLE_TYPE appear together with related data elements that provide starting location and applicable length information for specific data fields. Within all PDS data object definitions, the byte, bit, and record positions are counted from left to right, or first to last encountered, beginning with 1.

Data values may be represented within data files as ASCII or BINARY format. The ASCII storage format is simpler to transfer between different hardware systems and often between different application programs on the same computer. However, strictly numeric data often are stored in binary numeric types, since the ASCII representation of most numeric values requires more storage space than does the binary format. For example, each 8-bit pixel value in an image file would require 3 bytes if stored in ASCII format.

3.1 Data Elements

Table 3.1 identifies the data elements that provide data type, location, and length information according to the objects in which they appear.

3.2 Data Types

Table 3.2 identifies the valid values that may appear for the DATA_TYPE, BIT_DATA_TYPE, and SAMPLE_TYPE data elements (or their aliases) in PDS data object definitions. Many of the values in this table have been aliased to other values. Providing aliases allows the PDS to support and maintain backward compatibility. However, the preferred method is to use the value rather than its alias.

Unless noted as ASCII, all values in the table are binary.

Table 3.1: Data-Type-Related Elements Used in Data Label Objects

Data Object	**Data Elements**	**Notes**
COLUMN (without ITEMS)	DATA_TYPE	
	START_BYTE	
	BYTES	
COLUMN (with ITEMS)	DATA_TYPE	ITEM_TYPE is an alias
	START_BYTE	
	BYTES (opt)	total bytes in COLUMN
	ITEMS	
	ITEM_BYTES	size for each ITEM
BIT_COLUMN (without ITEMS)	BIT_DATA_TYPE	
	START_BIT	
	BITS	
BIT_COLUMN (with ITEMS)	START_BIT	
	BITS (opt)	Total bits in BIT_COLUMN
	ITEMS	
	ITEM_BITS	size for each ITEM
IMAGE	SAMPLE_TYPE	
	SAMPLE_BITS	
HISTOGRAM	DATA_TYPE	ITEM_TYPE is alias
	BYTES (opt)	total bytes in HISTOGRAM
	ITEMS	
	ITEM_BYTES	size for each ITEM (bin)

Table 3.2:   PDS Standard Data Types

Data Element Usage Codes:

D	=	DATA_TYPE
B	=	BIT_DATA_TYPE
S	=	SAMPLE_TYPE

Data Element **Usage**	**Value**	**Description**
D	ASCII_REAL	ASCII character string representation of real number
D	ASCII_INTEGER	ASCII character string representation of integer
D	ASCII_COMPLEX	ASCII character string representation of complex
D	BIT_STRING	alias for MSB_BIT_STRING
D, B	BOOLEAN	True/False indicator; 1, 2, or 4 byte unsigned number or 1-32 bit number; all 0's False; anything else True
D	CHARACTER	any ASCII character string
	COMPLEX	alias for IEEE_COMPLEX
D	DATE	ASCII character string representation of PDS date
D	EBCDIC_CHARACTER	any EBCDIC character string
	FLOAT	alias for IEEE_REAL
D	IBM_COMPLEX	IBM 360/370 mainframe complex number (8,16 byte)
D	IBM_INTEGER	IBM 360/370 mainframe 1, 2, and 4 byte numbers
D	IBM_REAL	IBM 360/370 mainframe real number (4 and 8 byte)
D	IBM_UNSIGNED_INTEGER	IBM 360/370 mainframe 1, 2, and 4 byte numbers
D	IEEE_COMPLEX	includes 8, 16, and 20 byte complex numbers
D, S	IEEE_REAL	includes 4, 8 and 10 byte real numbers
D	INTEGER	Single byte integers only
	INTEGER	alias for MSB_INTEGER (2+ bytes)
D	LSB_BIT_STRING	includes 1, 2, and 4 byte columns containing bit columns
D, S	LSB_INTEGER	includes 1, 2, and 4 byte numbers
D, S	LSB_UNSIGNED_INTEGER	includes 1, 2, and 4 byte numbers
	MAC_COMPLEX	alias for IEEE_COMPLEX
	MAC_INTEGER	alias for MSB_INTEGER
	MAC_REAL	alias for IEEE_REAL
	MAC_UNSIGNED_INTEGER	alias for MSB_UNSIGNED_INTEGER
D	MSB_BIT_STRING	includes 1, 2, and 4 byte columns containing bit columns

D, B	MSB_INTEGER	includes 1, 2, and 4 byte numbers
D, B, S	MSB_UNSIGNED_INTEGER	includes 1, 2, and 4 byte numbers, and 1-32 bit numbers
D, B	N/A	Used for spare (or unused) fields, if identified
D	PC_COMPLEX	includes 8, 16, 20 byte complex numbers
	PC_INTEGER	alias for LSB_INTEGER
D	PC_REAL	includes 4, 8, and 10 byte real numbers
	PC_UNSIGNED_INTEGER	alias for LSB_UNSIGNED_INTEGER
	REAL	alias for IEEE_REAL
	SUN_COMPLEX	alias for IEEE_COMPLEX
	SUN_INTEGER	alias for MSB_INTEGER
	SUN_REAL	alias for IEEE_REAL
	SUN_UNSIGNED_INTEGER	alias for MSB_UNSIGNED_INTEGER
D	TIME	ASCII character string representation of PDS date/time
	UNSIGNED_INTEGER	alias for MSB_UNSIGNED_INTEGER (2+bytes)
D, B, S	UNSIGNED_INTEGER	single byte numbers, or 1-32 bit numbers
	VAX_BIT_STRING	alias for LSB_BIT_STRING
D	VAX_COMPLEX	includes D, F, and H type complex numbers
	VAX_DOUBLE	alias for VAX_REAL
	VAX_INTEGER	alias for LSB_INTEGER
D, S	VAX_REAL	includes D (8 byte), F (4 byte), and H (16 byte) type real numbers
	VAX_UNSIGNED_INTEGER	alias for LSB_UNSIGNED_INTEGER
D	VAXG_COMPLEX	G type complex numbers only
D	VAXG_REAL	G type (8 byte) real numbers only

3.3 Binary Integers

There are two widely used formats for integer representations in 16-bit and 32-bit binary fields. These are the most-significant-byte first (MSB) and least-significant-byte first (LSB) architectures. The MSB architectures are used on IBM mainframes, many UNIX minicomputers (SUN, Apollo) and Macintosh computers. The LSB architectures are used on VAX systems and IBM PCs. The default interpretation for PDS labeled data is the MSB architecture, and non-specific data types (e.g. UNSIGNED_INTEGER) are aliased to MSB types. Therefore, files written on VAX or IBM PC hosts must specify LSB data types for binary integer fields, or use the appropriate aliases.

3.4 Signed versus Unsigned

The PDS default binary integer is a signed value in 2's complement notation. Therefore, a data type specified as INTEGER is interpreted as a signed integer. Unsigned binary integers must be identified using a valid UNSIGNED_INTEGER data type from Table 3.2.

3.5 Floating Point Formats

The PDS default representation for floating point numbers is in ANSI/IEEE standard. This representation is defined as the PDS IEEE_REAL data type, and aliases are identified in Table 3.2. Several specific floating point representations are supported by PDS, and are further described in Appendix C.

3.6 Bit String Data

A BIT_STRING data type is used for COLUMNs to hold individual bit field values. Each bit field is defined in a BIT_COLUMN object. A BIT_STRING data type can be a 1, 2, or 4 byte field, much like a binary integer. Extraction of specific bit fields within a 2 or 4 byte BIT_STRING is dependent on the host architecture (MSB or LSB), and follows the binary integer specifications identified in Section 3.3 above. In interpreting bit fields (BIT_COLUMNS) within a BIT_STRING, any necessary conversions (byte swapping from LSB to MSB) are done first, and then bit field (START_BIT, BITS) values are used to extract the appropriate bits. This will assure that bit fields are not fragmented due to differences in hardware architectures.

3.7 Format Specifications

The data format specification is used to determine the format for display of a data value. The following FORTRAN data format specifications will be used:

Aw	Character data value.
Iw	Integer value.
Fw.d	Floating point value, displayed in decimal format.
Ew.d[Ee]	Floating point value, displayed in exponential format.

Where:

w=	Total number of positions in the output field (including sign, decimal point or “E”).
d=	Number of positions to the right of the decimal point.
e=	Number of positions in exponent length field.

3.8 Internal Representations of Data Types

Appendix C contains the detailed internal representation of the PDS standard data types listed in Table 3.2.

PDS has developed tools that are designed to use the specifications outlined in Appendix C for interpreting data values for display and validation.

Chapter 4

Data Products

A data product is a grouping of primary and secondary data objects and their associated PDS labels resulting from a scientific observation. Three examples of a data product are a PDS labeled image, a spectrum, and a time series table. A data product is a component of a data set (see the *Data Set/ Data Set Collection Contents and Naming* chapter of this document).

Each data product is made up of one or more primary data objects, secondary data objects, and PDS data product labels.

Primary Data Object

A Primary data object is a grouping of data results from a scientific observation. The actual science data, such as an image or table, represents the measured instrument parameters.

Secondary Data Object

A Secondary data object is any data needed for processing or interpreting the primary data object. Each primary data object may have one or more associated secondary data objects. An example of a secondary data object is a histogram derived from an image.

A PDS data product label, expressed in Object Description Language (ODL) (see the *Object Description Language (ODL) Specification and Usage* chapter of this document), identifies, describes and defines the structure of the data. There may be a single label to describe the data product, or separate labels for each data object.

4.1 Data Product File Configurations

The grouping of primary and associated secondary data objects and their PDS label(s) into one or more physical files can be done in a variety of ways. An important consideration in choosing a file organization scheme for a data product is the intended use of the PRODUCT_ID data element. The PRODUCT_ID uniquely identifies an individual data product and can be based on physical file names.

Example:

An image (the data product in this example) is a color triplet having three primary data objects, stored in separate physical files, one for each of the red, blue, and green images. Each is uniquely identified by a PRODUCT_ID, additionally they are logically associated through the IMAGE_ID data element.

for the red image:

```
        PRODUCT_ID              = "22A190-RED"
        IMAGE_ID                = "22A190"
```

for the blue image:

```
        PRODUCT_ID            = "22A190-BLUE"
        IMAGE_ID              = "22A190"
```

for the green image:

```
        PRODUCT_ID            = "22A190-GREEN"
        IMAGE_ID              = "22A190"
```

Figure 4.1 illustrates file configurations for a data product with a single data object.

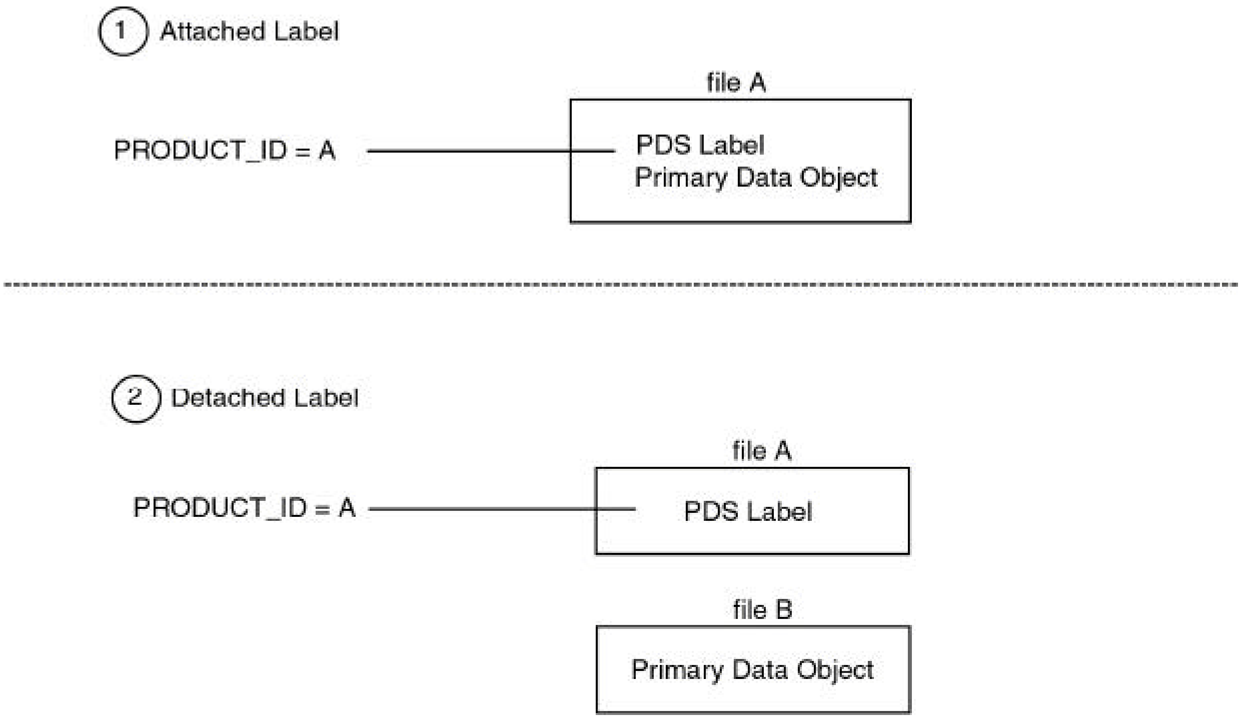


Figure 4.1 Data Product with a Single Data Object

For a data product having multiple data objects (one or more primary data objects and one or more secondary data objects), the assignment of the PRODUCT_ID is identified within the label of the data product file(s).

Figure 4.2 shows five possible file configurations for a single data product that consists of two data objects, a primary and secondary data object. Similar examples could be made using data products composed of several primary data objects.

Note that the use of options (2) and (4) would require a logical linking by another identification data element in each label.

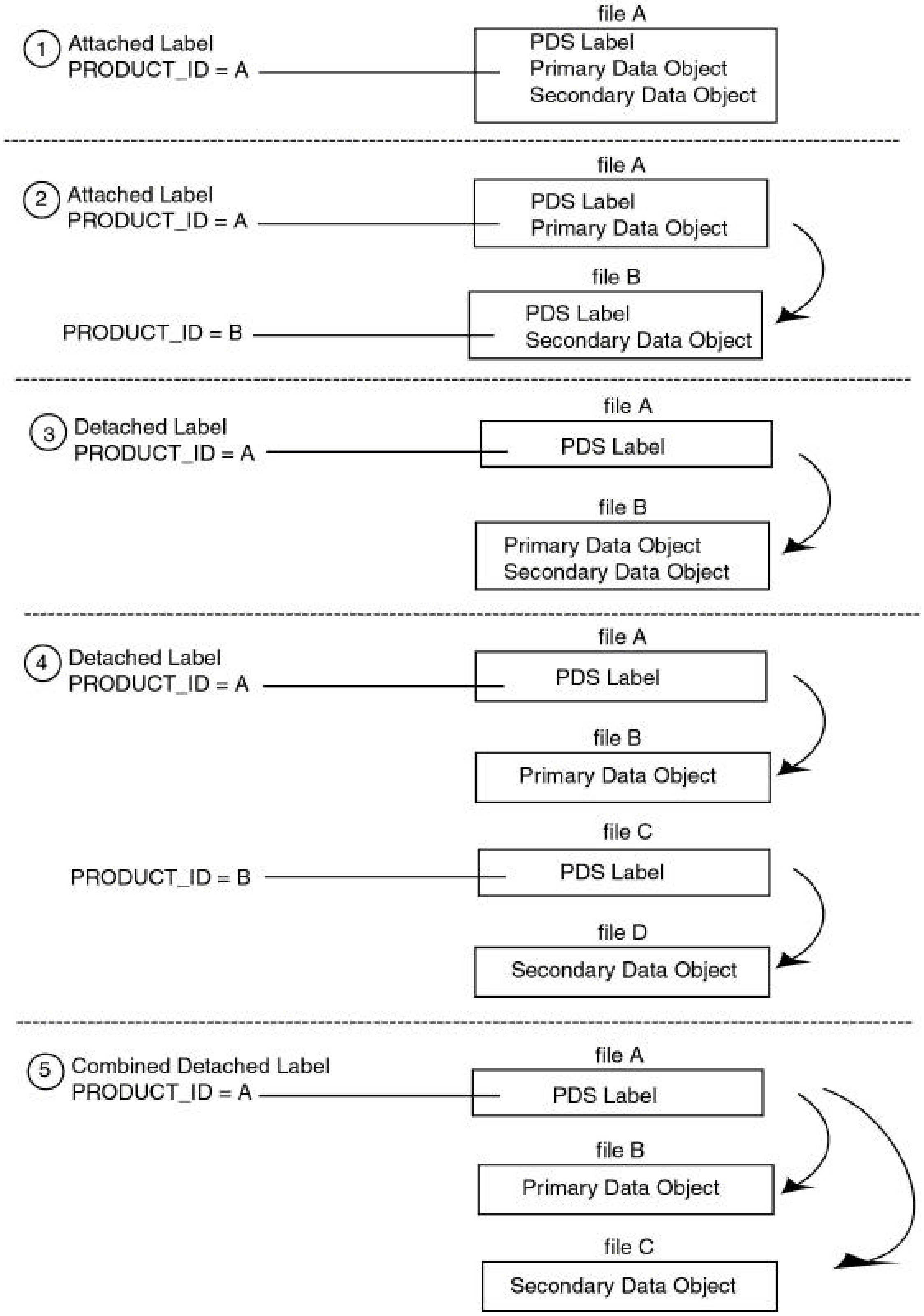


Figure 4-2. Data Product with Multiple Data Objects

Chapter 5

Data Product Labels

PDS data product labels are required for describing the contents and format of each individual data product within a data set. PDS data product labels are written in the Object Description Language (ODL). The PDS has chosen to label the wide variety of data products under archival preparation by implementing a standard set of data object definitions, data elements, and standard values for the elements. These data object definitions, data elements, and standard values are defined in the *Planetary Science Data Dictionary* (PSDD). Appendix A of this document provides general descriptions and examples of the use of these data object definitions and data elements for labeling data products.

5.1 Format of PDS Labels

5.1.1 Labeling methods

In order to identify and describe the organization, content, and format of each data product, PDS requires a distinct data product label for each individual data product file. These distinct product labels may be constructed in one of three ways:

Attached - The PDS data product label is embedded at the beginning of the data product file. There is one label attached to each data product file.

Detached - The PDS data product label is detached from the data and resides in a separate file which points to the data product file. There is one detached label file for every data product file. The label file should have the same base name as its associated data file, but the extension .LBL .

Combined Detached - A single PDS detached data product label file is used to describe the contents of more than one data product file. The combined detached label points to individual data products.

NOTE: Although all three labeling methods are equally acceptable, the PDS tools do not currently support the Combined Detached label option.

Figure 5.1 illustrates the use of each of these methods for labeling individual data product files.

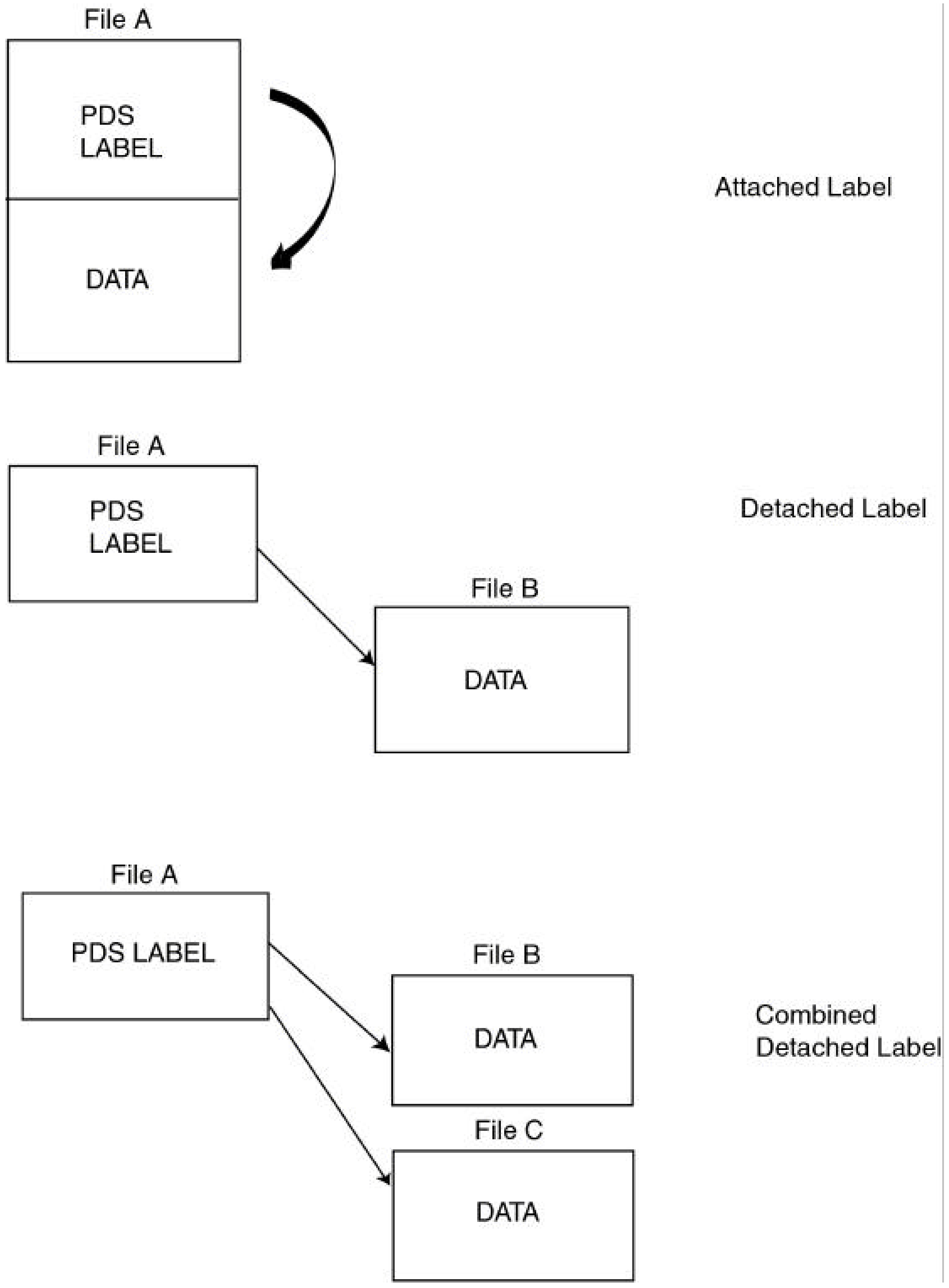


Figure 5.1 Attached, Detached, and Combined Detached PDS Labels

5.1.2 Label format

PDS recommends that labels have stream record format, and line lengths of at most 80 characters (inclusive of the CR/LF line terminators) so that the entire label can be seen on a computer screen without horizontal scrolling. The Carriage Return and Line Feed (CR/LF) pair is the required line terminators for all PDS labels. (See the *Record Formats* chapter of this document.)

All values in a PDS label should be in upper case, except values for description fields. It is also recommended that the equal signs in the labels be aligned for ease of reading.

ASCII Character Set

All values in a PDS label must conform to the standard ASCII character set. These values include the range of characters 001 through 127 (decimal). The character set 128-255 is not used in PDS because these characters vary by font / symbol set and are not predictable when viewed and/or printed. Similarly, TAB characters are not to be used as they are interpreted differently by different applications. Each TAB character represents ’n’ number of SPACE characters and ’n’ will vary by application thereby causing nicely formatted tables/columns to become misaligned.

Label Padding

For a fixed length data file with an attached label, the label is padded with SPACEs (ASCII decimal 32) in one of the following ways:

1) Spaces are added after the label’s END <CR><LF> statement and before the data so that the total size of the label is an integral multiple of the record length of the data.

Example:
In the example below, the label portion of the file is 7 x 324 = 2268 bytes in length, including blank fill between the END<CR><LF> statement and the first byte of data. The actual data portion of the file starts at record 8 (i.e., the 1st byte of the 8th record starts at byte (7 x 324)+1 = 2269)

```
RECORD_TYPE                    = FIXED_LENGTH<CR><LF>
RECORD_BYTES                   = 324<CR><LF>
FILE_RECORDS                   =  334<CR><LF>
LABEL_RECORDS                  =  7<CR><LF>

^IMAGE                         =  8<CR><LF>

END<CR><LF>
....blank fill....
data
```

2) Each line in the label may be padded with SPACEs so that each line in the label has the same record length as the data file. In this case, the label line length may exceed the recommended 80 characters.

Example:
In the example below, the label portion of the file is 80 x 85 = 6800 bytes in length. Each line in the label portion of the file is 85 bytes long, the same length as each data record. Notice the blank space between the actual values in the label and the line delimiters. In the example, the label is 80 lines long (i.e., 80 records long) and the data begins at record 81. Note that the label is padded so that <CR><LF> are in bytes 84 and 85.

```
RECORD_TYPE                     = FIXED_LENGTH       <CR><LF>
RECORD_BYTES                    = 85                 <CR><LF>
FILE_RECORDS                    = 300                <CR><LF>
LABEL_RECORDS                   = 80                 <CR><LF>
...
^TABLE                          = 81                 <CR><LF>
END                                                  <CR><LF>
data
```

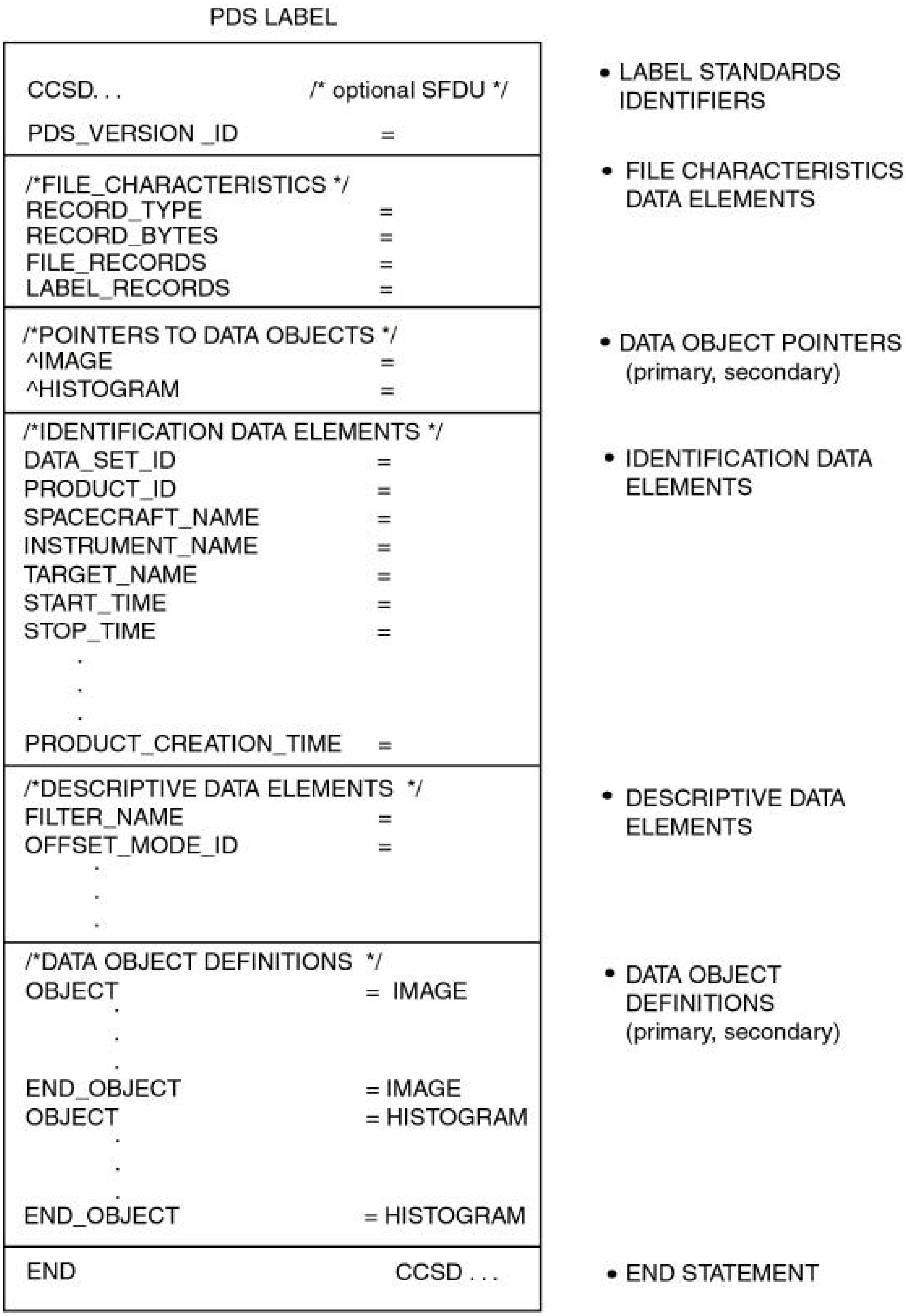
5.2 Data Product Label Content

5.2.1 Attached and Detached Labels

PDS data product labels have a general structure that is used for all attached and detached labels, except for data products described by minimal labels. (Minimal labels are described in Section 5.2.3.)

- LABEL STANDARDS identifier
- FILE CHARACTERISTIC data elements
- DATA OBJECT pointers
- IDENTIFICATION data elements
- DESCRIPTIVE data elements
- DATA OBJECT DEFINITIONS
- END statement

Figure 5.2 provides an example of how this general structure appears in an attached or detached label for a data product file containing multiple data objects.



```
PDS LABEL

CCSD. . .                          /* optional SFDU */
PDS_VERSION _ID                    =

/*FILE_CHARACTERISTICS */
RECORD_TYPE                        =
RECORD_BYTES                       =
FILE_RECORDS                       =
LABEL_RECORDS                      =

/*POINTERS TO DATA OBJECTS */
^IMAGE                             =
^HISTOGRAM                         =

/*IDENTIFICATION DATA ELEMENTS */
DATA_SET_ID                        =
PRODUCT_ID                         =
SPACECRAFT_NAME                    =
INSTRUMENT_NAME                    =
TARGET_NAME                        =
START_TIME                         =
STOP_TIME                          =
      .
      .
      .
PRODUCT_CREATION_TIME              =

/*DESCRIPTIVE DATA ELEMENTS  */
FILTER_NAME                        =
OFFSET_MODE_ID                     =
        .
        .
        .

/*DATA OBJECT DEFINITIONS  */
OBJECT                             =  IMAGE
          .
          .
          .
END_OBJECT                         = IMAGE
OBJECT                             = HISTOGRAM
          .
          .
          .
END_OBJECT                         = HISTOGRAM

END                                CCSD . . .
```

• LABEL STANDARDS IDENTIFIERS
• FILE CHARACTERISTICS DATA ELEMENTS
• DATA OBJECT POINTERS (primary, secondary)
• IDENTIFICATION DATA ELEMENTS
• DESCRIPTIVE DATA ELEMENTS
• DATA OBJECT DEFINITIONS (primary, secondary)
• END STATEMENT

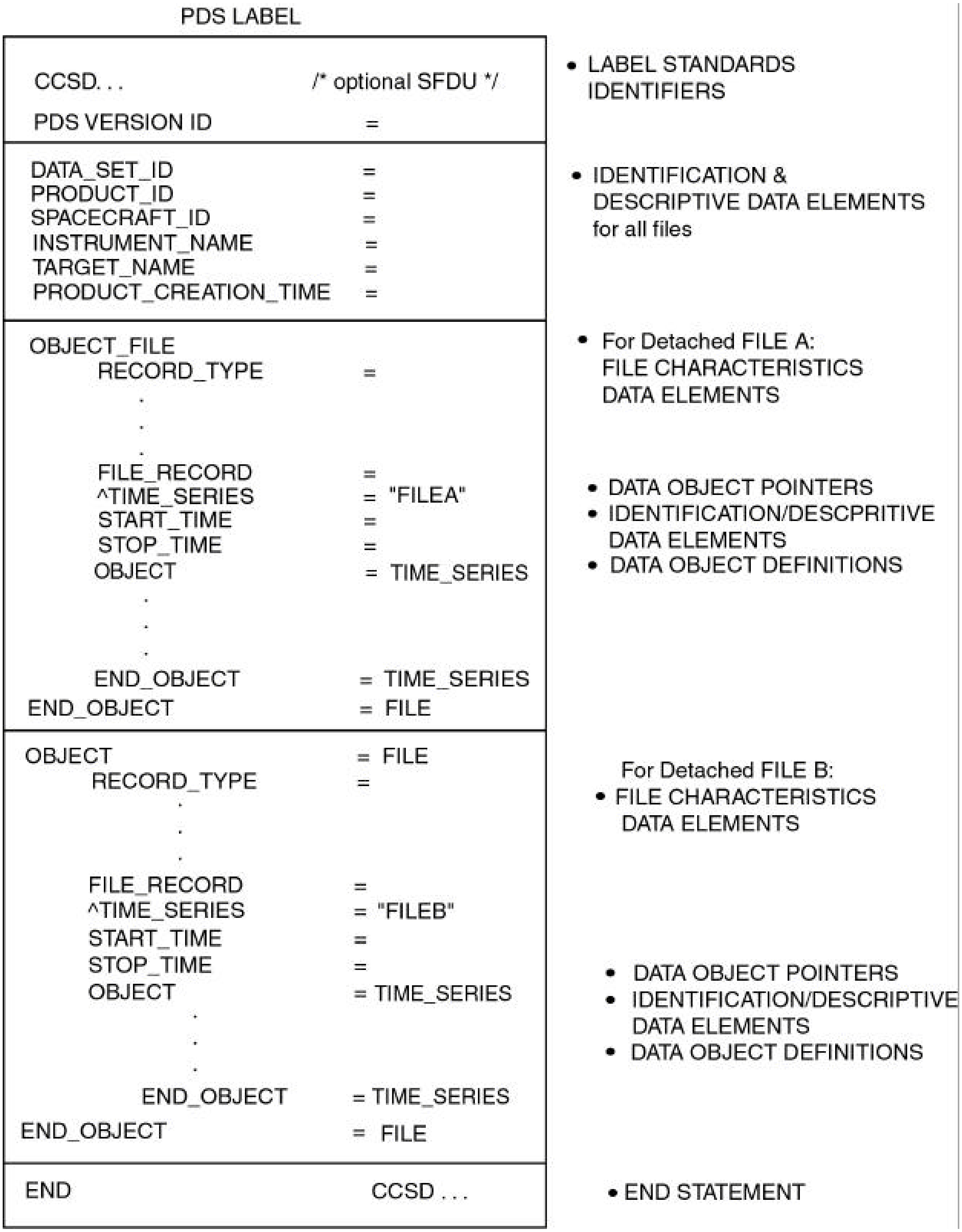
Figure 5.2 PDS Attached / Detached Label Structure

5.2.2 Combined Detached Labels

For the Combined Detached label option, the general label structure is modified slightly to explicitly reference each individual file within its own FILE object. In addition, identification and descriptive data elements that apply to all of the files can be located before the FILE objects.

- LABEL STANDARDS identifiers
- IDENTIFICATION data elements that apply to all referenced data files
- DESCRIPTIVE data elements that apply to all referenced data files
- OBJECT=FILE statement (Repeats for each data product file)
    - FILE CHARACTERISTIC data elements
    - DATA OBJECT pointers
    - IDENTIFICATION data elements
    - DESCRIPTIVE data elements
    - DATA OBJECT DEFINITION
- END_OBJECT=FILE statement
- END statement

Figure 5.3 provides an example of how this general structure appears in a combined detached label that describes more than one data product file.



PDS LABEL

```
CCSD. . .                          /* optional SFDU */
PDS VERSION ID                      =

DATA_SET_ID                         =
PRODUCT_ID                          =
SPACECRAFT_ID                       =
INSTRUMENT_NAME                     =
TARGET_NAME                         =
PRODUCT_CREATION_TIME               =

OBJECT_FILE
        RECORD_TYPE                 =
             .
             .
             .
        FILE_RECORD                 =
        ^TIME_SERIES                =  "FILEA"
        START_TIME                  =
        STOP_TIME                   =
        OBJECT                      =  TIME_SERIES
             .
             .
             .
        END_OBJECT                  =  TIME_SERIES
END_OBJECT                          =  FILE

OBJECT                              =  FILE
        RECORD_TYPE                 =
                .
                .
                .
        FILE_RECORD                 =
        ^TIME_SERIES                =  "FILEB"
        START_TIME                  =
        STOP_TIME                   =
        OBJECT                      =  TIME_SERIES
                 .
                 .
                 .
             END_OBJECT             =  TIME_SERIES
END_OBJECT                          =  FILE

END                                    CCSD . . .
```

- LABEL STANDARDS IDENTIFIERS

- IDENTIFICATION & DESCRIPTIVE DATA ELEMENTS for all files

- For Detached FILE A: FILE CHARACTERISTICS DATA ELEMENTS

- DATA OBJECT POINTERS
- IDENTIFICATION/DESCPRITIVE DATA ELEMENTS
- DATA OBJECT DEFINITIONS

For Detached FILE B:
- FILE CHARACTERISTICS DATA ELEMENTS

- DATA OBJECT POINTERS
- IDENTIFICATION/DESCRIPTIVE DATA ELEMENTS
- DATA OBJECT DEFINITIONS

- END STATEMENT

Figure 5.3 PDS Combined / Detached PDS Label Structure

5.2.3 Minimal Labels

Use of the minimal label option is only allowed when the format of the data cannot be supported by the current documented Data Objects.

For minimal labels, the general label structure has removed the required use of data objects. A minimal label does not contain any PDS data object definitions or pointers to data objects. The above applies to both attached and detached labels.

Minimal labels must satisfy the following requirements:

(1) Provide the ability to locate the data (file) associated with the label.

1a. Attached labels

Since data objects and pointers are not required in the minimal label, by definition the data follows immediately after the label.

1b. Detached Labels

Both the implicit and explicit use of the FILE object are supported. The FILE_NAME keyword, contained in the FILE object, is required.

(2) Provide the ability to locate a description of the format/content of the data. One of the following must be provided in the minimal label:

2a. ^DESCRIPTION = “<filename>”
This is a pointer to a file containing a detailed description of the data format; may be located in the same directory as the data or in the DOCUMENT subdirectory.

2b. DESCRIPTION = “<text appears here>”
This is either a detailed description of the data file, its format, data types,and use, or it is a reference to a document available externally, e.g., a Software Interface Specification (SIS) or similar document.

(3) When minimal labels are used, DATA_OBJECT_TYPE = FILE should be used in the Data Set Catalog template. If a situation arises where multiple Data Object types are used, then
separate Data Set Catalog templates should be provided for each data product. And, where appropriate, use a Data Set Collection template.

5.2.3.1 Implicit File Object (Attached and Detached Minimal Label)

The general structure for minimal labels with implicit file objects is as follows:

- LABEL STANDARDS identifier

- FILE CHARACTERISTIC data elements
- IDENTIFICATION data elements
- DESCRIPTIVE data elements
- END statement

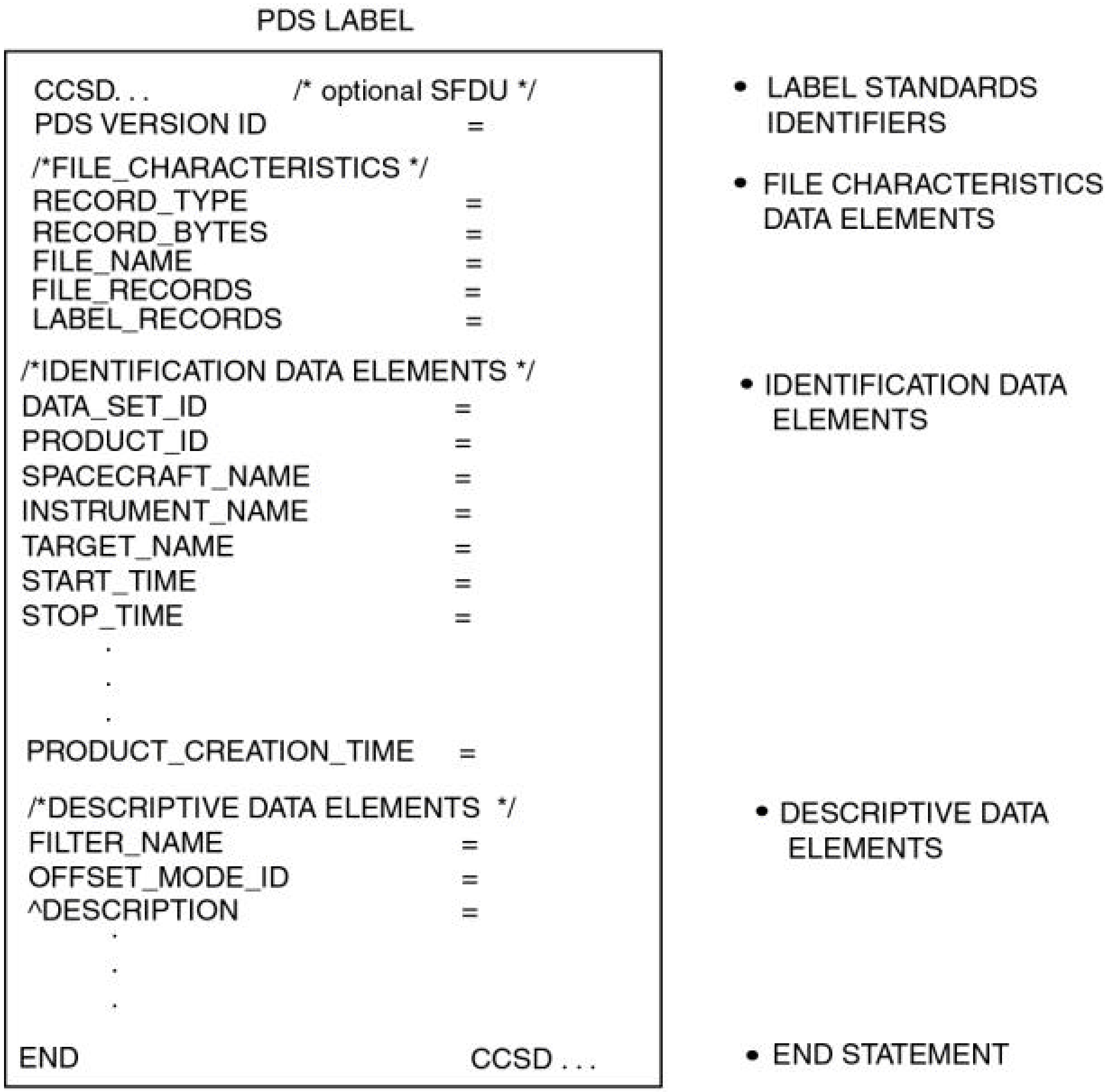
5.2.3.2 Explicit File Object (Detached Minimal Label)

The general structure for minimal labels with explicit file objects is as follows:

- LABEL STANDARDS identifier
- IDENTIFICATION data elements
- DESCRIPTIVE data elements
- OBJECT=FILE statement
    - FILE CHARACTERISTIC data element

- END_OBJECT=FILE
- END statements

Figure 5.4 provides an example of how this general structure appears in a detached minimal label. In this example, an implicit FILE object is used.



PDS LABEL

```
CCSD. . .              /* optional SFDU */
PDS VERSION ID                    =
/*FILE_CHARACTERISTICS */
RECORD_TYPE                       =
RECORD_BYTES                      =
FILE_NAME                         =
FILE_RECORDS                      =
LABEL_RECORDS                     =
/*IDENTIFICATION DATA ELEMENTS */
DATA_SET_ID                       =
PRODUCT_ID                        =
SPACECRAFT_NAME                   =
INSTRUMENT_NAME                   =
TARGET_NAME                       =
START_TIME                        =
STOP_TIME                         =
        .
        .
        .
PRODUCT_CREATION_TIME             =
/*DESCRIPTIVE DATA ELEMENTS  */
FILTER_NAME                       =
OFFSET_MODE_ID                    =
^DESCRIPTION                      =
        .
        .
        .
END                                  CCSD . . .
```

- LABEL STANDARDS IDENTIFIERS
- FILE CHARACTERISTICS DATA ELEMENTS
- IDENTIFICATION DATA ELEMENTS
- DESCRIPTIVE DATA ELEMENTS
- END STATEMENT

Figure 5.4 PDS Detached Minimal Label Structure

5.3 Detailed Label Contents Description

This section describes the detailed requirements for the content of PDS labels. The subsections describe label standards identifiers, file characteristic data elements, data object pointers, identification data elements, descriptive data elements, data object definitions, and the END statement.

5.3.1 Label Standards Identifiers

Each PDS label begins with an optional Standard Formatted Data Unit (SFDU) label and a PDS_VERSION_ID data element:

```
CCSD....                [optional SFDU label]
PDS_VERSION_ID
```

The PDS does not require SFDU labels on individual products, but they may be needed for conformance with specific project or other agency requirements. If SFDUs are provided on a data product, they must follow the standards described in the *SFDU Usage* chapter in this document.

The PDS requires the PDS_VERSION_ID data element to identify the PDS published standards that the label adheres to. This version id will be used to provide PDS software tool support for a specific set of standards and will allow the evolution and expansion of both standards and tools as required by the PDS user community.

For labels adhering to the standards described in this document -- the *PDS Standards Reference*, Version 3.3 -- and its associated *Planetary Science Data Dictionary*, Version 3.0, this will be:

```
PDS_VERSION_ID = PDS3
```

5.3.2 File Characteristic Data Elements

PDS data product labels contain data element information that describes important attributes of the physical structure of a data product file. The PDS file characteristic data elements are:

```
RECORD_TYPE
RECORD_BYTES
FILE_RECORDS
LABEL_RECORDS
```

The RECORD_TYPE data element identifies the record characteristics of the data product file. A complete discussion of the RECORD_TYPE data element and its use in describing data products produced on various platforms is provided in the *Record Formats* chapter in this document. The RECORD_BYTES data element identifies the number of bytes in each physical record in the data product file. The FILE_RECORDS data element identifies the number of physical records in the file. The LABEL_RECORDS identifies the number of physical records that make up the PDS product label.

Not all of these data elements are required in every data product label. Table 5.1 lists the required (Req) and optional (Opt) file characteristic data elements for a variety of data products and labeling methods for both attached (Att) and detached (Det) labels. Where (max) is specified, the value indicates the maximum size of any physical record in the file.

Table 5.1: File Characteristic Data Element Requirements

Labeling Method	Att	Det	Att	Det	Att	Det	Att	Det
RECORD_TYPE	FIXED_LENGTH		VARIABLE_LENGTH		STREAM		UNDEFINED	
RECORD_BYTES	Req	Req	Rmax	Rmax	Omax	-	-	-
FILE_RECORDS	Req	Req	Req	Req	Opt	Opt	-	-
LABEL_RECORDS	Req	-	Req	-	Opt	-	-	-

Note: For detached minimal labels, the FILE_NAME keyword is required.

5.3.3 Data Object Pointers

The actual data whose structure and attributes are defined in a PDS label are “data objects”. Each data product file may contain one or more data objects.

The PDS uses a pointer mechanism within product labels to identify the starting locations for all primary and secondary data objects in a data product.

Examples of primary data objects that may require data object pointers include:

TABLE
IMAGE
SERIES
SPECTRUM
QUBE

Examples of secondary data objects that may require data object pointers include:

HISTOGRAM
PALETTE
HEADER
DOCUMENT

5.3.3.1 Use of Pointers in Attached Labels

For attached labels, if there is only one data object referenced, a data object pointer is not required. However, it is strongly recommended that data object pointers be used at all times. The data object is assumed to start in the next physical record after the PDS product label area. This is commonly the case with ASCII text files described by a TEXT object and ASCII SPICE files described by a SPICE_KERNEL object. The top two illustrations in Figure 5.5 show example files that do not require data object pointers.

If multiple data objects are stored in the data product file, object pointers are required for all data objects. The syntax for data object pointers in attached labels may take one of two forms:

^<object_identifier> = nnn (see the *Object Description Language* chapter in this document)

where nnn represents the starting record number within the file,
Or,

^<object_identifier> = nnn <BYTES>

where nnn represents the starting byte location within the file.

The following two illustrations in Figure 5.5 show the required use of data object pointers for attached label products containing multiple data objects.

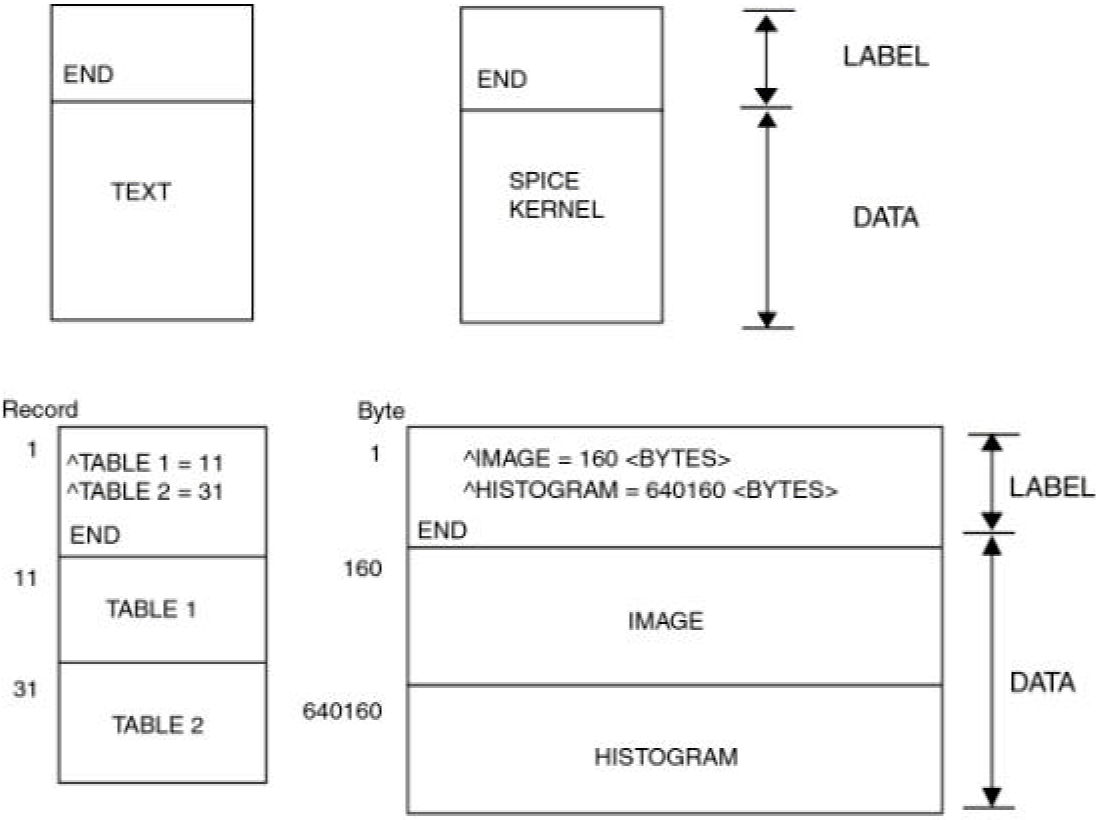


Figure 5.5 Data Object Pointers-Attached Labels

5.3.3.2 Use of Pointers in Detached and Combined Detached Labels

If the PDS data product label is a detached or a combined detached label, data object pointers are required for all data objects referenced.

The syntax for data object pointers may take one of three forms:

(1) ^*object_identifier* = “filename”
(2) ^*object_identifier* = (“filename”, nnn)
(3) ^*object_identifier* = (“filename”, nnn <BYTES>)

With respect to these three cases, these object pointers reference either byte or record locations in data files that are detached, or separate from, the label file.

In each case, the filename is the name of the file containing the data object and is detached, or separate from, the label file. In the first case, the data object is located at the beginning of the referenced file. In the second case, the data object begins nnn physical records from the beginning of the referenced file. In the third case, the data object begins nnn bytes from the beginning of the referenced file.

```
^QUBE                 = ("DATA.DAT")
^ENGINEERING_TABLE    = ("DATA.DAT", 10)
^TABLE                = ("DATA.DAT", 10 <BYTES>)
```

Figure 5.6 illustrates several examples of data object pointer usage for data product files with detached or combined detached labels. The top example shows a data product consisting of a HEADER data object and a TABLE data object together in a single file. The detached label for this product includes pointers for both data objects, with the TABLE object starting at byte 601 of file A. The middle example illustrates a combined detached label for a data product contained in two data objects, each in a separate file. A separate pointer is provided for each data object. The bottom example shows a detached label for a data product containing multiple data objects.

Where multiple data objects are stored within a data product file, and where multiple data objects occupy portions of the same physical record, the data object pointer indicates the first physical record containing the data object. Additional data elements within the Data Object Definitions (e.g. LINE_PREFIX_BYTES, ROW_SUFFIX_BYTES) provide the relative byte locations within each record for each line or row of data within the data object.

5.3.3.3 Note Concerning Minimal Attached and Detached Labels

By definition, data object pointers do not exist in minimal labels. The format of the data is fully described in a separate file or document.

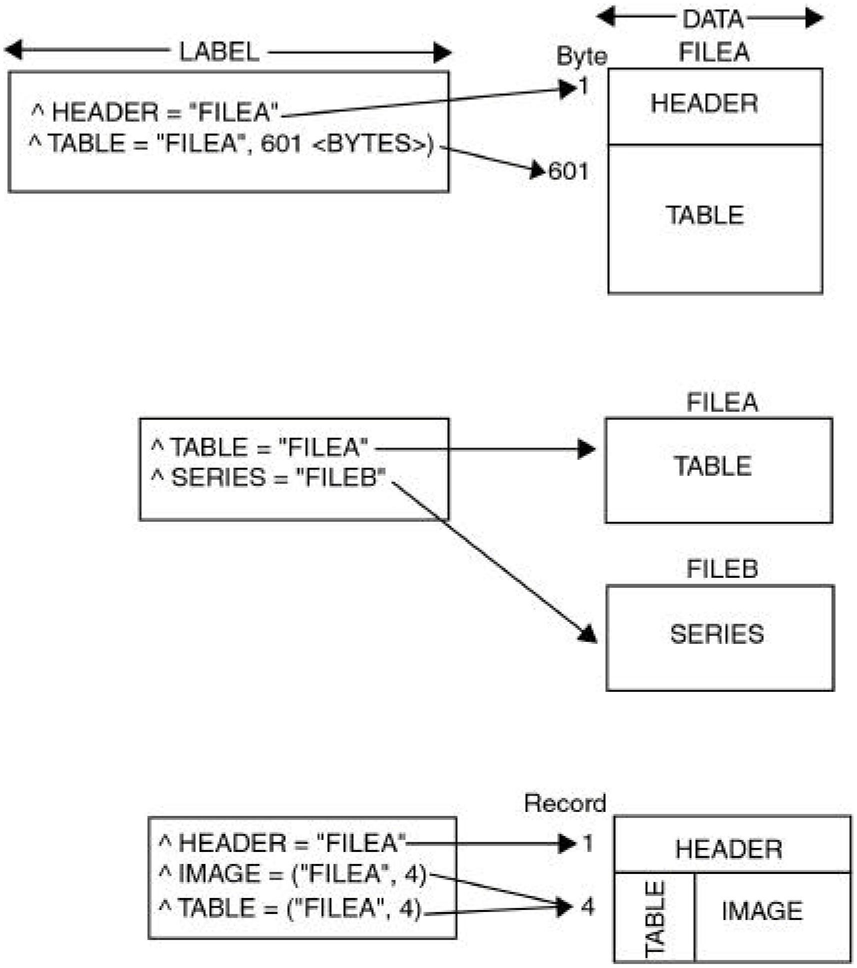


Figure 5.6 Data Object Pointers – Detached & Combined Labels

5.3.4 Identification Data Elements

The identification data elements provide important information about the data to uniquely identify the data product and to associate it with other data products that may be related. This information is often used to populate the PDS product level catalogs or inventories. PDS requires a minimum set of these identification data elements to be included in all product labels. These requirements vary depending on the type of data product being archived. Additional identifying data elements may be required by specific projects or organizations.

Additional data elements which might be needed to further identify the data objects or which would be needed to catalog the data product to support potential search criteria should also be included. These additional data elements are selected from the *Planetary Science Data Dictionary* (PSDD).

NOTE: When a data element is needed for a data product label, but is not yet recorded in the PSDD, it can be proposed to be added to the dictionary. See a PDS Data Engineer for assistance.

5.3.4.1 Spacecraft Science Data Products

The following identification data elements shall be included in data product labels for all spacecraft science data products:

DATA_SET_ID
PRODUCT_ID
INSTRUMENT_HOST_NAME
INSTRUMENT_NAME
TARGET_NAME
START_TIME
STOP_TIME
SPACECRAFT_CLOCK_START_COUNT
SPACECRAFT_CLOCK_STOP_COUNT
PRODUCT_CREATION_TIME

5.3.4.2 Earthbased Science Data Products

The following identification data elements shall be included in data product labels for all earthbased science and radio science data products:

DATA_SET_ID
PRODUCT_ID
INSTRUMENT_HOST_NAME
INSTRUMENT_NAME
TARGET_NAME
START_TIME
STOP_TIME
PRODUCT_CREATION_TIME

5.3.4.3 Ancillary Data Products

The following identification data elements shall be included in data product labels for all ancillary data sets. These types of products may be more general in nature, supporting a wide variety of instruments for a particular mission. For example, SPICE data sets, general engineering data sets, and uplink data are considered ancillary data products.

DATA_SET_ID
PRODUCT_ID
PRODUCT_CREATION_TIME

The following data elements are highly recommended, and should be included in ancillary data products whenever they apply:

INSTRUMENT_HOST_NAME
INSTRUMENT_NAME
TARGET_NAME
START_TIME
STOP_TIME
SPACECRAFT_CLOCK_START_COUNT
SPACECRAFT_CLOCK_STOP_COUNT

5.3.5 Descriptive Data Elements

In addition to the identification data elements required for various types of data, PDS strongly recommends including additional data elements related to specific types of data. These descriptive data elements must include any data elements which might be needed to interpret or process the data objects or which would be needed to catalog the data product to support potential search criteria at the product level.

Not only will these values be available with the data to the user, but they are also used to load PDS product level catalogs and inventories with descriptive information about each data product. PDS product level catalogs and inventories at PDS Discipline Nodes support both online data product access and ordering capabilities.

In addition, PDS has developed and continues to develop software display and analysis packages for standard data objects. These software packages will be built to utilize various descriptive data elements.

Recommendations for descriptive data elements to consider supplying will come from working with PDS Mission Interface personnel as well as the data producer's own suggestions. These additional data elements are selected from the *Planetary Science Data Dictionary*.

NOTE: When a data element is needed for a data product label, but is not yet recorded in the PSDD, it can be proposed to be added to the dictionary. See the PDS Data Engineer for assistance with submitting new data elements for inclusion in the PSDD.

Pointers are sometimes used in a PDS label to provide a shorthand method for including a set of descriptive data elements (e.g., ^DESCRIPTION) or a long descriptive text passage referenced in several data product labels.

5.3.6 Data Object Definitions

The PDS requires a separate data object definition within a product label for describing the structure and associated attributes of each data object in the data product. There will be one data object definition for every primary and secondary data object pointer identified in Section 5.2.3. These data object definitions are of the form:

```
OBJECT           = aaa        where aaa is the name of the data object
...
END_OBJECT       = aaa
```

The PDS has designed a set of standard data object definitions to be used for labeling data products. Among these standard objects are those designed to describe data structures commonly used for scientific data storage. Appendix A provides a complete set of PDS data object definition requirements, along with examples of data product labels.

Pointers are sometimes used in a PDS label to provide a shorthand method for including a set of data sub-objects referenced in several data product labels. For example, a ^STRUCTURE is often used to include a set of COLUMN sub-objects for a TABLE structure that is used in many labels.

NOTE: Minimal labels do not contain any data object definitions.

5.3.7 End Statement

The end of the PDS label is identified by the required END statement followed by an optional SFDU.

The PDS does not require SFDU labels on individual products, but they may be required to conform with specific project or other agency requirements. If SFDUs are provided on a data product, they must follow the standards described in the *SFDU Usage* chapter in this document. In some, but not all cases, another SFDU label is required after the PDS END statement to provide “end label” and sometimes “start data” information.

Chapter 6

Data Set/Data Set Collection Contents and Naming

One of the objectives of the PDS is to introduce consistency in the contents, organization and naming of planetary data sets. The PDS has introduced the concept that an archive quality data set collection or data set must include everything that is needed to understand and utilize the data. Towards this goal, the PDS has worked with the PDS Discipline Nodes, numerous Flight Projects, individual scientists and programmers, and the NSSDC to develop approaches to ensure that this consistency is achieved.

Figure 6.1 shows the relationships between Data Set Collection, Data Sets, and Data Products. Figure 6.2 shows the logical and physical relationships.

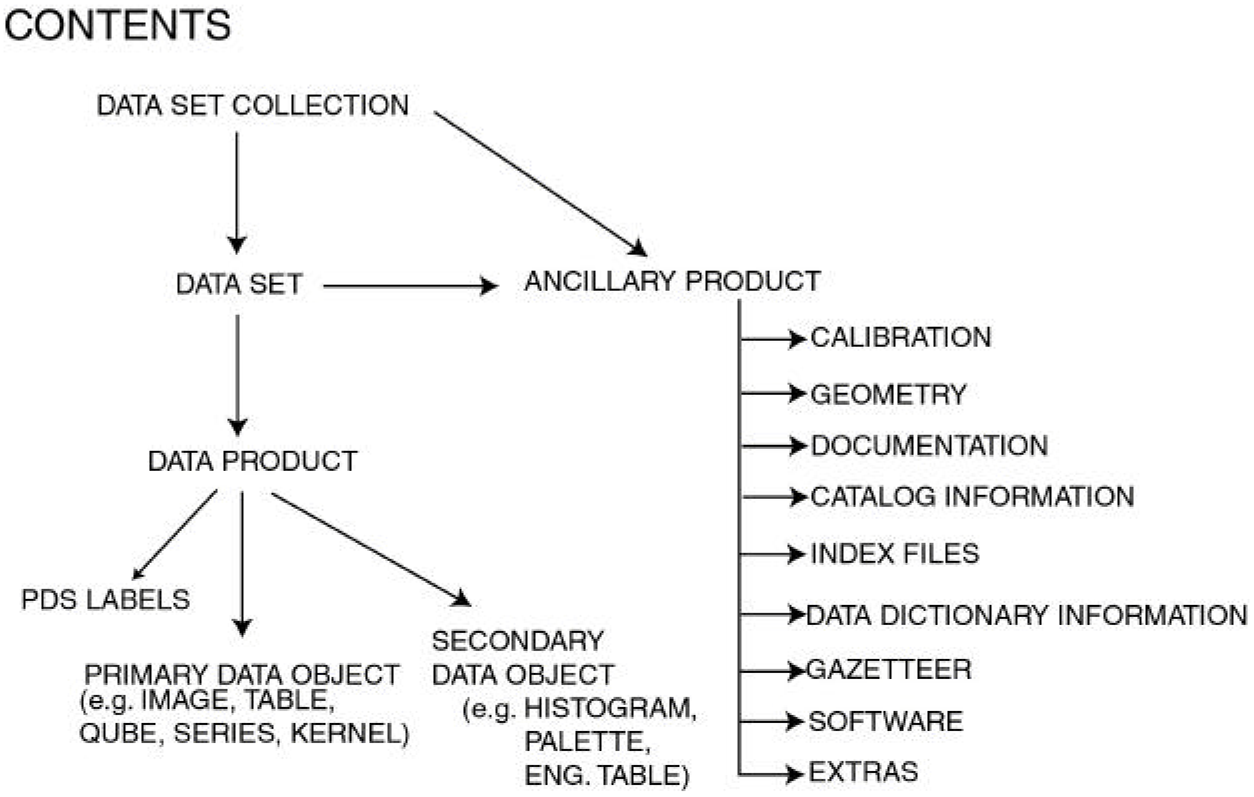


Figure 6.1 Data Set Collection, Data Sets, and Data Products

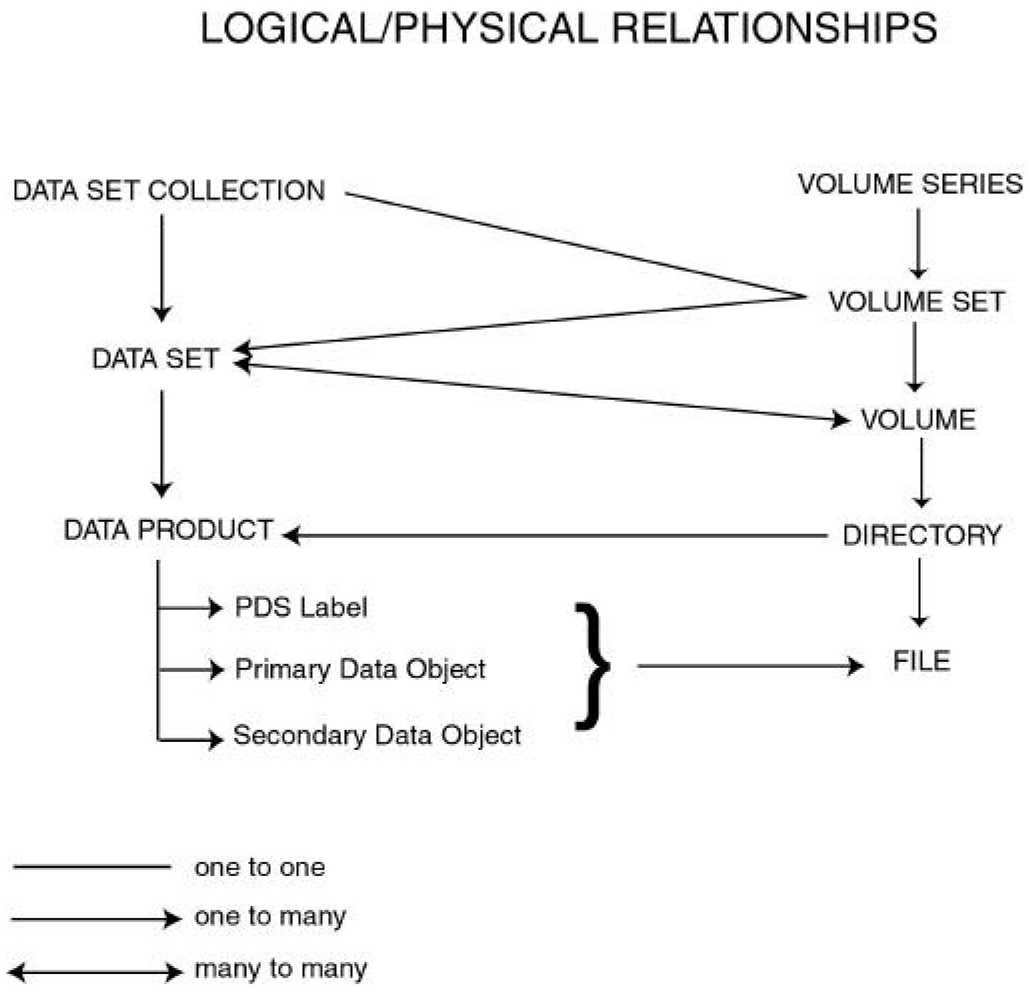


Figure 6.2 Logical and Physical Relationships of Data Products

6.1 Data Set/Data Set Collection Contents

Data Set Collection and Data Set defined:

Data Set Collection - A data set collection consists of data sets that are related by observation type, discipline, target, or time, and therefore are to be treated as a unit, to be archived and distributed as a group (set) for a specific scientific objective and analysis. One of the primary considerations in creating a data set collection is that the collection as a whole provides more utility than any individual data set within the collection. Further, the individual data set(s) may be ineffective unless used in concert with the other data sets in the collection.

Data Set - The accumulation of data products, secondary data, software, and documentation, that completely document and support the use of those data products. A data set can be part of a data set collection.

A data set collection or a data set may include all of the following:

Data Product - A labeled grouping of data resulting from a scientific observation. Examples of data products include planetary images, spectrum tables, and time series tables. A data product is a component of a data set.

Calibration data - Calibration files used in the processing of the raw data or needed to use the data.

Geometry data - Relevant files (e.g., SEDRs, SPICE kernels) needed to describe the observation geometry.

Documentation - All the textual material which describes the mission, spacecraft, instrument, and data set. This can include references to science papers, or the actual papers.

Catalog Information - High-level descriptive information about a data set (e.g. mission description, spacecraft description, instrument description), expressed in Object Description Language (ODL) which is suitable for loading into a catalog.

Index Files - Information which allows the user to locate the data of interest, such as a table mapping latitude/longitude ranges to file names.

Data Dictionary Information - A portable version of the *Planetary Science Data Dictionary* which is pertinent to the data set. The dictionary is expressed in ODL.

Gazetteer - Information about the named features on a target body associated with the data sets.

Software- The software libraries, utilities, or application programs to access/process the data objects.

All data sets submitted to the PDS shall include the software used and/or algorithms for original data reduction, processing, calibration and, decalibration of the data, or documentation stating how to obtain such software. When software accompanies a data set, the source code, build instructions, and software documentation shall be included.

There are several other types of data set software which may be provided with a data set:

1. Special software which is developed and maintained for certain hardware platforms. This is often a refined version of the processing software developed for mission data analysis.

2. Utilities which allow a user to select parameters from the data set and to extract these parameter values to a data file based on certain key values (event time, for example). The output format should be a simple ASCII table or one of the other generic PDS data object formats. This is a minimum level of access for conducting a peer review of a data set.

3. Data analysis tools such as plotting programs.

6.2 Data Set Naming and Identification

This standard contains instructions for naming a PDS data set and forming a Data Set Identifier. Every PDS data set shall be given a DATA_SET_NAME and DATA_SET_ID, both formed from seven components. All components are required except for the Data Set Type and Description components. These components are described in section 6.4.

The only characters allowed within a data_set_id are the upper case alphanumeric set (A-Z, 0-9), a forward slash (/), a period (.), and a hyphen (-). The period is only used with numerics, i.e., V1.0 or 12.5SEC. No other special characters are allowed (e.g., underscore (_)).

Multiple instrument hosts, instruments, or targets shall be referenced in a DATA_SET_NAME or DATA_SET_ID by concatenation of the values with a forward slash (/) which is interpreted as "and."

The data set identifier (DATA_SET_ID) shall not exceed 40 characters in length. Each component shall be the acronym rather than a full length name used in forming the DATA_SET_NAME. Within the data_set_id, acronyms shall be separated by hyphens.(See section 6.4 for valid acronyms.)

A DATA_SET_NAME shall not exceed 60 characters in length. Where the character limitation is not exceeded, the full length name of each component should be used. If the full length name is too long, an acronym shall be used to abbreviate components of the name. (See section 6.4 for valid full length names and acronyms.)

The intent of the data set name (DATA_SET_NAME) and identifier (DATA_SET_ID) is primarily to uniquely identify the data set.

The components of the DATA_SET_NAME and DATA_SET_ID are:

Instrument host
Target
Instrument
Data processing level number
Data set type (optional)
Description (optional)
Version number

Example:

- Full length data set name: Mariner 9 and Viking Orbiter 1 and Viking Orbiter 2 Mars Imaging Science Subsystem and Visual Imaging Subsystem derived cloud data Version 1.0

- DATA_SET_NAME = “MR9/V01/V02 MARS IMAGING SCIENCE SUBSYSTEM/VIS 5 CLOUD V1.0”

- DATA_SET_ID = “MR9/V01/V02-M-ISS/VIS-5-CLOUD-V1.0”

In this example, Instrument hosts are Mariner 9, Viking Orbiter 1 and Viking Orbiter 2
Target is Mars
Instruments are the Imaging Science Subsystem and Visual Imaging Subsystem
Data Processing Level number is 5
Description is CLOUD
Version number is V1.0
The optional Data set type is not used in this example.

6.3 Data Set Collection Naming and Identification

This standard contains instructions for naming a PDS data set collection and forming an identifier. A data set collection consists of data sets that are related by observation type, discipline, target, or time (which are treated as a unit), for a specific scientific purpose.

A data set collection will contain data sets that may cover several targets, be of different processing levels, and have different instrument hosts and instruments. Since the individual data sets will be identified by their own data set names, some of this information is not necessary to repeat at the collection level. Therefore, the DATA_SET_COLLECTION_NAME uses a subset of the DATA_SET_NAME components in addition to a new component, collection name, which identifies the group of related data sets.

The DATA_SET_COLLECTION_NAME and DATA_SET_COLLECTION_ID are formed from the six components listed below. All are required, except for data processing level number, data set type, and description. However, it is recommended that data set type or description be used whenever possible.

The only characters allowed within a DATA_SET_COLLECTION_ID are the upper case alphanumeric set (A-Z, 0-9), a forward slash (/), a period (.), and a hyphen (-). The period is only used with numerics, i.e., V1.0 or 12.5SEC. No other special characters are allowed (e.g., underscore (_)).

Multiple targets or data processing levels shall be referenced in the data set collection name or identifier by concatenation of the values with a forward slash (/) which is interpreted as “and.”

A DATA_SET_COLLECTION_NAME shall not exceed 60 characters in length. Where the character limitation is not exceeded, the full length name of each component should be used. If the full length name is too long, an acronym shall be used to abbreviate it. (See Section 6.4 for valid full length names and acronyms.)

The DATA_SET_COLLECTION_ID shall not exceed 40 characters in length. Each component shall be the acronym rather than a full length name used in forming the DATA_SET_COLLECTION_NAME. Within the DATA_SET_COLLECTION_ID, acronyms shall be separated by hyphens. (See Section 6.4 for valid acronyms.)

The components of the DATA_SET_COLLECTION_NAME and DATA_SET_COLLECTION_ID are:

Collection name
Target
Data processing level number (optional)
Data set type (optional)
Description (optional)
Version number

Example:
The Pre-Magellan Data Set Collection contains radar and gravity data similar to the kinds of data that Magellan collected and was used for pre-Magellan analyses of Venus and for comparisons to actual Magellan data.

• Full-length data set collection name: Pre-Magellan Earth, Moon, Mercury, Mars, and Venus Resampled and Derived Radar and Gravity Data Version 1.0

• DATA_SET_COLLECTION_NAME = “PRE-MAGELLAN E/L/H/M/V 4/5 RADAR / GRAVITY DATA V1.0”

• DATA_SET_COLLECTION_ID = “PREMGN-E/L/H/M/V-4/5-RAD/GRAV-V1.0”

6.4 Description of Name and ID Components

If the information needed to describe your data is not listed, consult the PDS Data Engineer to determine what the appropriate acronyms are for you to use.

When a reference is made to the PSDD, see the standard values list for the data elements.

1. **Instrument host** component valid values are:

full length names:	INSTRUMENT_HOST_NAME data element in the PSDD
acronyms:	INSTRUMENT_HOST_ID data element in the PSDD
exceptions:	for Earth based data sets with no instrument host defined, the default value of EAR is recommended.

2. **Collection name** component valid values may be one of the following:

GRSFE	Geological Remote Sensing Field Experiment
IHW	International Halley Watch
PREMGN	Pre-Magellan

3. **Target** component valid values are:

full length names: TARGET_NAME data element in the PSDD
acronyms: one of the following target IDs

Target ID	Target Name
A	Asteroid
C	Comet
CAL	Calibration
D	Dust
E	Earth
H	Mercury
J	Jupiter
L	Moon
M	Mars
MET	Meteorite
N	Neptune
P	Pluto
R	Ring
S	Saturn
SA	Satellite
SS	Solar System
U	Uranus
V	Venus
X	Other, ex. Checkout
Y	Sky

NOTE: Satellites or rings shall be referenced in a DATA_SET_NAME and DATA_SET_ID by the concatenation of the satellite or ring identifier with the associated planet identifier; for example:

JR = Jupiter's rings
JSA = Jupiter's satellites

If Jupiter data are also included in the ring and/or satellite data set, then only Jupiter, J, is referenced as the target.

In cases where there are data sets of comets or asteroids this component represents the TARGET_TYPE rather than the target name, for example:

A = Asteroid	CAL = Calibration
C = Comet	MET = Meteorite

Valid values for the TARGET_TYPE data element are found in the PSDD.

4. **Instrument** component valid values are:

full length names:	INSTRUMENT_NAME data element in the PSDD
acronyms:	INSTRUMENT_ID data element in the PSDD
exceptions:	ENG or ENGINEERING for engineering data sets SPICE for SPICE data sets GCM for Global Circulation Model data SEDR for supplemental EDR data POS for positional data

5. **Data processing level number**

This component is the National Research Council (NRC) Committee on Data Management and Computation (CODMAC) data processing level number.

Normally a data set contains data of one processing level. PDS recommends that data of different processing levels be treated as different data sets. However, if it is not possible to separate the data, then a single data set with multiple processing levels will be accepted. Use the following when specifying the data processing level number component of the data set identifier and name:

(a) the processing level number of the largest subset of data or
(b) the highest processing level number if there is no predominant subset.

DATA LEVEL NUMBER (CODMAC AND NASA LEVELS)

Level	Proc. Type	Data Processing Level Description
1	Raw Data	Telemetry data with data embedded.
2	Edited Data	Corrected for telemetry errors and split or decommutated into a data set for a given instrument. Sometimes called Experimental Data Record. Data are also tagged with time and location of acquisition. Corresponds to NASA Level 0 data.
3	Calibrated Data	Edited data that are still in units produced by instrument, but that have been corrected so that values are expressed in or are proportional to some physical unit such as radiance. No resampling, so edited data can be reconstructed. NASA Level 1A.
4	Resampled Data	Data that have been resampled in the time or space domains in such a way that the original edited data cannot be reconstructed. Could be calibrated in addition to being resampled. NASA Level lB.
5	Derived Data	Derived results, as maps, reports, graphics, etc. NASA Levels 2 through 5.
6	Ancillary Data	Nonscience data needed to generate calibrated or resampled data sets. Consists of instrument gains, offsets; pointing information for scan platforms, etc.
7	Correlative Data	Other science data needed to interpret spaceborne data sets. May include ground-based data observations such as soil type or ocean buoy measurements of wind drift.
8	User Description	Description of why the data were required, any peculiarities associated with the data sets, and enough documentation to allow secondary user to extract information from the data.
N	N	Not Applicable

6. **Data set type**

Normally, the data processing level (CODMAC) component is sufficient to be able to identify the type or level of data. However, if additional identification is desired, this component may be used. The following is a list of valid values (both full length names and acronyms) that may be used for this component.

NOTE: Several of the values in this table are currently unique to a particular mission (e.g. BIDR, MIDR were used on Magellan). These values should also be used on other missions, if deemed appropriate.

Acronym	Description
ADR	Analyzed Data Record
BIDR	Basic Image Data Record
CDR	Composite Data Record
CK	SPICE CK (Pointing Kernel)
DDR	Derived Data Record (possibly multiple instruments)
DIDR	Digitalized Image Data Record
DLC	Detailed Level Catalog
EDC	Existing Data Catalog
EDR	Experiment Data Record
EK	SPICE EK (Instrument Kernel)
GDR	Global Data Record
IDR	Intermediate Data Record
IK	SPICE IK (Instrument Kernel)
LSK	SPICE LSK (Leap Second Kernel)
MDR	Master Data Record
MIDR	Mosaicked Image Data Record
ODR	Original Data Record
PCK	SPICE PCK (Planetary Constants Kernel)
PGDR	Photograph Data Record
RDR	Reduced Data Record
REFDR	Reformatted Data Record
SDR	System Data Record

SEDR	Supplementary Experiment Data Record
SPK	SPICE SPK (Ephemeris Kernel)
SUMM	Summary (data) (to be used in the browse function)
SAMP	Sample data from a data set (not subsampled data)

7. **Description**

The following is a list of example values (both full length names and acronyms) that could be used for this component.

While the description is optional, it allows the user to provide information to help describe the data set, such as identifying a specific comet or asteroid.

Acronym	Description
ALT/RAD	Altimetry and Radiometry
BR	Browse
CLOUD	Cloud
ELE	Electron
ETA-AQUAR	Meteor Eta-Aquarius
FULL-RES	Full Resolution
GIACOBIN-ZIN	Comet Giacobini Zinner
HALLEY	Comet Halley
ION	Ion
LOS	Line of Sight Gravity
MOM	Moment
PAR	Parameter
SA	Spectrum Analyzer
SA-4.0SEC	Spectrum Analyzer 4.0 second
SA-48.0SEC	Spectrum Analyzer 48.0 second

8. Version number

The rules for determining version numbers for PDS Data Sets/Data Set Collections are as follows:

(a) If there is not a previous version of the PDS data set/data set collection, then use Version 1.0.

(b) If a previous version exists, then consider the following:

i. If the data sets/data set collections contain the same set of data, but use a different medium (e.g., CD-ROM), then no new version number is required (i.e. no new data set identifier). The inventory system will handle the different media for the same data set.

ii. If the data sets/data set collections contain the same set of data, but have minor corrections or improvements such as a change in descriptive labeling, then the version number is incremented by a tenth. For example, V1.0 becomes V1.1.

iii. If a data set/data set collection has been reprocessed, using, for example, a new processing algorithm or different calibration data, then the version number is incremented by one (V1.0 would become V2.0). Also, if one data set/data set collection contains a subset, is a proper subset, or is a superset of another, then the version number is incremented by one.

Chapter 7

Date/Time Format

PDS has adopted a subset of the International Standards Organization Standard (ISO/DIS) 8601 standard entitled "Data Element and Interchange Formats - Representations of Dates and Times", and applies the standard across all disciplines in order to give the system generality. See also Dates and Times in *Object Description Language* (Chapter 12, Section 12.3.2) of this document.

It is important to note that the ISO/DIS 8601 standard covers only ASCII representations of dates and times.

7.1 Date/Times

In the PDS there are two date/time formats recognized as legal:

CCYY-MM-DDTHH:MM:SS.sssZ (preferred format)
CCYY-DDDTHH:MM:SS.sssZ

Each format represents a concatenation of the conventional date and time expressions with the two parts separated by the letter T:

CC	-	century (00-99)
YY	-	year (00-99)
MM	-	month (01-12)
DD	-	day of month (01-31)
DDD	-	day of year (001-366)
T	-	date/time separator
HH	-	hour (00-23)
MM	-	minute (00-59)
SS	-	second (00-59)
sss	-	fractions of second (000-999)

The time part of the expression represents time in Universal Time Coordinated (UTC), hence the Z at the end of the expression (see Section 7.3.1 for further discussion). Note that in both the PDS Data Set Catalog and data product labels the "Z" is optional and is assumed.

The preferred date/time format both for labels and Data Set Catalog templates is CCYY-MM-DDTHH:MM:SS.sssZ.

Date/Time Precision

The above date/time formats can be truncated to match the precision of the date/time value. The following are examples of date/time values having limited precision:

1998
1998-12
1998-12-01
1998-12-01T23
1998-12-01T23:59
1998-12-01T23:59:58
1998-12-01T23:59:58.1

ODL Date/Time Information

Chapter 12, Object Description Language (ODL) Specification and Usage, section 12.3.2, Dates and Times, of this document provides additional information on the use of ODL in date/time formation, representation, and implementation.

7.2 Dates

The PDS allows dates to be expressed in conventional and native (alternate) formats.

7.2.1 Conventional Dates

Conventional dates shall be represented as either year, month, and day-of-month or as year and day-of-year using the full ISO/DIS 8601 format, which has the fields separated by dash characters, as follows: CCYY-MM-DD or CCYY-DDD. Both formats are acceptable for use in PDS labels and Data Set Catalog templates, but the PDS recommends the CCYY-MM-DD convention.

7.2.2 Native Dates

The format of a native date is user specified. An example of a native date is Julian Day, an integer count of days since a given reference day (January 1, 4713 B.C.)

7.3 Times

The PDS allows times to be expressed in conventional and native (alternate) formats.

7.3.1 Conventional Times

Conventional times shall be represented as hours, minutes, and seconds using the full ISO/DIS 8601 format. The hours, minutes, and seconds consist of three two-digit fields separated by

colons, with the field values being modulo 24, 60, and 60, respectively. The seconds field may be optionally followed by a fractional part; if fractions of seconds are specified, a period shall be used as the decimal point and not the European-style comma. The fractional part shall be at most 3 digits long.
The PDS has adopted the use of Universal Time Coordinated (UTC) for expressing time, using the format HH:MM:SS.sssZ. Note that in both the PDS Data Set Catalog and data product labels the “Z” is optional and is assumed. Fractions of seconds cannot exceed a precision of milliseconds.

The START_TIME and STOP_TIME data elements required in data product labels and catalog templates use the UTC format. For data collected by spacecraft-mounted instruments, the date/ time shall be a time which corresponds to “spacecraft event time”. For data collected by instruments not located on a spacecraft, this time shall be an earth-based event time value.

Adoption of UTC (rather than spacecraft-clock-count, for example) as the standard facilitates comparison of data from a particular spacecraft or ground-based facility with data from other sources.

7.3.2 Native Times

Native or alternate time formats may be represented in a data product label or Data Product Catalog using the NATIVE_START_TIME and NATIVE_STOP_TIME elements. Native times also can be represented using specific data elements. Such data elements may be proposed by the data supplier and reviewed by the PDS.

The following paragraphs describe typical examples of native time formats.

1. **Spacecraft Clock Count (sclk)** - Spacecraft clock count (sclk) provides a more precise time representation than event time for instrument-generated data sets, and so may be desirable as an additional time field. In a typical instance, a range of spacecraft-clock-count values (i.e., a start-and a stop-value) may be required.

   Spacecraft clock count (SPACECRAFT_CLOCK_START_COUNT and SPACECRAFT_
   CLOCK_STOP_COUNT) shall be represented as a right-justified character string field with a maximum length of thirty characters. This format will accommodate the extra decimal point appearing in these data for certain spacecraft and other special formats, while also supporting the need for simple comparison (e.g., “>” or “<”) between clock count values.

2. **Longitude of Sun** - Longitude of Sun (“L sub S”) is a derived data value which can be computed, for a given target, from UTC.

3. **Ephemeris Time** - Ephemeris time (ET) is calculated as “TAI + 32.184 sec. + periodic terms”. The NAIF S and P kernels have data that are in ET, but the user (via NAIF ephemeris readers which perform data conversion) can obtain the UTC values.

4. **Relative Time** - In addition to event times, certain “relative time” fields will be needed to represent data times or elapsed times. Time-from-closest-approach is an example of such a data element. These times shall be presented in a (D,H,M,S) format as a floating point number, and should include fractional seconds when necessary. The inclusion of “day” in relative times is motivated by the possible multi-day length of some delta times, as could occur, for example, in the case of the several-month Galileo Jupiter orbit.

5. **Local Times** - For a given celestial body, LOCAL_TIME is the hour relative to midnight in units of 1/24th the length of the solar day for the body.

6. **Alternate Time Zones (Relative to UTC)** - When times must be expressed according to an alternate time zone, they shall consist of hours, minutes, seconds, and an offset, in the form HH:MM:SS.sss+n, where n is the number of hours from UTC.

Chapter 8

Directory Types and Naming

The Directory Naming standard defines the convention for naming subdirectories on a data volume. This standard lists the predetermined, standard directories that have been established by PDS, plus the rules for forming subdirectory names and abbreviations.

8.1 Standard Directory Names

When any of the following directories are included on an archive product, the following standard directory naming convention must be used.

CATALOG — Template subdirectory containing PDS Catalog templates.

DOCUMENT — Documentation subdirectory containing supplemental / ancillary material which augments the understanding or use of the data products.

EXTRAS — Subdirectory to house "value added" elements of the volume beyond the scope of the PDS required elements.

GAZETTER — Gazetteer subdirectory containing tables of information about the geological features of a target.

INDEX — Data and inventory index subdirectory containing files which allow users to locate data of interest.

LABEL — Label subdirectory containing include files which describe the data format and organization.

SOFTWARE — Software subdirectory containing utilities, application programs, or subprograms used to access or process data files.

The following standard directory names are recommended for use on archive volumes.

CALIB — Calibration data subdirectory containing calibration files used in original processing of data, or needed to use the data.

GEOMETRY — Geometry data subdirectory containing relevant files (SEDRs, spice kernels) needed to describe observation geometry.

BROWSE — Data subdirectory containing reduced resolution version of data products.

DATA — A single subdirectory containing one or more data subdirectories each of which contains data products. The DATA subdirectory should be used to unclutter the root directory of multiple data directories.

Note that some data sets may not contain all the components above and, as a result, do not need all of the directories listed above. See the *Volume Organization and Naming* chapter of this document for the required and optional subdirectories on a volume. For example, many image data sets do not include geometry files and so do not need a GEOMETRY directory.

8.2 Formation of Directory Names

1. A directory name shall consist of capitalized alphanumeric characters and the underscore "_" character only (i.e., A-Z, 0-9, or "_"). No lowercase letters (i.e., a-z) or special characters (e.g., "#", "&", "*") are allowed.

2. A directory name shall not exceed 8 characters in length. The purpose of this is to comply with the ISO 9660 level 1 media interchange standard.

3. The first letter of a directory name shall be an alphabetic character, unless the directory name represents a year (e.g., 1984).

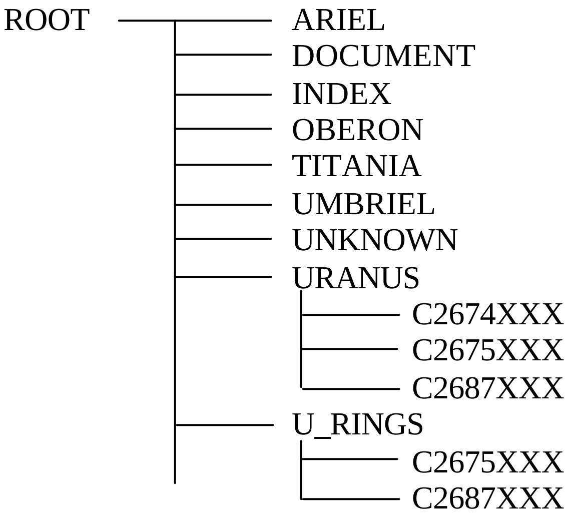
4. If numeric characters are used as part of the name (e.g., DIR1, DIR2, DIR3) the name should be padded with leading zeros up to the maximum size of the numeric part of the name (DIR0001, DIR0002, DIR3267).

5. Directories which contain a range of similarly named files shall be assigned directory names using the portion of the filename which encompasses all the files in the directory, with "X's" used to indicate the range of values of actual filenames in the directory.

   For example, the PDS Uranus Imaging CD-ROM disk contains image files which have filenames that correspond to SPACECRAFT_CLOCK_START_COUNT values. The directory that contains the image files ranging from C2674702.IMG through C2674959.IMG has the directory name C2674XXX/.

6. Directory names shall use full length terms whenever possible (e.g., SATURN, MAGELLAN, CRUISE, NORTH, DATA, SOFTWARE). Otherwise, directory names shall be constructed from abbreviations of full length names using the underscore character to separate abbreviated terms, if possible. The meaning of the directory name should be clear from the abbreviation and from the directory structure.

For example, the following directory structure can be found on the Voyager 2 Images of Uranus CD-ROM Volume 1:



In this case, it is clear from the context that the directory U_RINGS is the abbreviated form of URANUS_RINGS.

7. High level directories that deal with data sets covering a range of planetary science disciplines shall adhere to the following hierarchy:

A Planetary science directory: PLANET/
Planetary body subdirectories: MERCURY/, MOON/, MARS/, VENUS/, COMET/
Discipline subdirectories: ATMOS/, IONOSPHE/, MAGNETOS/, RING/, SURFACE/, and SATELLIT/
(Use satellite name if numerous files exist)

8. The recommended SOFTWARE subdirectory naming convention is described in the *Volume Organization and Naming* chapter of this document. A platform-based model or an application-based model can be used in defining software subdirectories. For a platform-based model, the hardware platform and operating system/environment must be explicitly stated. If there is more than one operating system/environment supported, then they must be subdirectories under the hardware directories. If there is only one, then the subdirectory can be promoted to the hardware directory.

For example, if software for the PC for both DOS and Windows were present on the volume, the directories SOFTWARE/PC/DOS and SOFTWARE/PC/WIN would exist.

If only DOS software were present, the directory would be SOFTWARE/PCDOS.

8.3 Path Formation Standard

The PDS standard for path names is based on Level 1 of the ISO 9660 International Standard. A pathname may consist of up to 8 directory levels. Each directory name shall be limited to 8 characters (A - Z, 0 - 9, _ (underscore)). PDS has also chosen the UNIX/POSIX forward slash separator (/) for use in path names. Path names typically appear on PDS volumes as data in index tables for locating specific files on an archive volume. They may also appear as values in a limited number of keywords (e.g. FILE_SPECIFICATION_NAME, PATH_NAME, and LOGICAL_VOLUME_PATH_NAME).

The following are examples of valid path names:

TG15NXXX/TG15N1XX/TG15N12X/ - identifies the location of the directory TG15N12X at the third level below the top level of an archive volume.

DOCUMENT/ - identifies a DOCUMENT directory within the root directory.

Note: The leading slash is omitted because these are relative paths. The trailing slash is included so that the concatenation of PATH_NAME and FILE_NAME gives the full file specification.

Previous PDS standards allowed the use of the DEC VMS syntax for path names. While PDS support for this format continues to exist, it is recommended that all future volumes shall use the UNIX syntax instead.

8.4 Tape Volumes

When magnetic tape is used as the archive medium, a directory structure cannot be used because the medium does not support multi-level directories. In this case, files must be stored in a sequential fashion, as if they were all located in the same directory.
A directory structure for the volume shall be designed in any case, so that when the data are transferred to a medium which supports hierarchical file structures, the data can then be placed into a multi-level directory structure. A DIRECTORY object shall be placed on each tape volume (within the VOLUME object) which is used to describe how the sequential files should be placed in a hierarchical structure.

8.5 Exceptions to These Standards

In certain cases, the archive media used to store the data, the hardware used to produce the data set, or the software which must operate on the data may impose restrictions on the names of directories and their overall organization. In these cases, the alternate directory organization and naming used on the data volume should be reviewed by PDS personnel during the data set submission process in order to determine the best compromise between the standard given above and any practical restrictions on the volume or data set structure.

Chapter 9

Documents

To improve utility of an archival product, supplemental and/or ancillary reference material (documents) may be included. This material is primarily textual and should augment, reinforce, and assist in the overall understanding and/or use of the data products or software. It may include, but is not limited to:

(1) flight project documents
(2) instrument papers
(3) science articles
(4) volume information
(5) software interface specifications (SISs)
(6) software user manuals

PDS has three guidelines for deciding whether to include documents: (1) Would this information be helpful to a data user? (2) Is the material necessary? (3) Is the documentation complete? Recognizing that there are many levels of inquiry regarding data sets, PDS seeks to err on the side of completeness.

Each document must be prepared and saved in a PDS compliant format, have a PDS compliant label, and be stored in the DOCUMENT directory of an archive volume (see the Volume Organization and Naming chapter of this document).

A readable ASCII text version of each document must be included on the archival volume; documents may also be provided in certain other PDS approved formats, at the option of the data producer. 'ASCII text' includes only printable ASCII characters and the ASCII blank space; it does not include special software constructs or graphics. 'Readable' means that special markup insertions do not significantly interfere with the user's ability to read the file. Plain ASCII text, HTML, TeX, and LaTeX files-subject to the 'readability' criterion-satisfy the 'ASCII text' PDS requirement.

9.1 PDS Objects for Documents

Either of two PDS objects-TEXT or DOCUMENT-may be used for documents, subject to the constraints in the next two subsections.

9.1.1 TEXT Objects

TEXT objects are preferred for documents which stand alone and have a narrow focus-for example, describing the contents of an archive volume (AAREADME.TXT) or the contents of a directory (DOCINFO.TXT). Each file which uses the PDS TEXT object must:

(a) have an attached or detached label containing a TEXT object definition (see Appendix A);

(b) be in plain, unmarked ASCII text (printable ASCII characters and the ASCII blank space), cannot include programming language constructs such as HTML, and cannot include graphics; and

(c) have a file name which ends with the extension .TXT.

9.1.2 DOCUMENT Objects

DOCUMENT objects are preferred when several versions of the same file are provided and/or when there are several component files in the document-for example, graphics in addition to text. Each file which uses the PDS DOCUMENT object must:

(a) have a detached label (or, in the case of a single file in plain ASCII text, optionally an attached label) containing a DOCUMENT object definition (see Appendix A);

(b) be in plain, unmarked ASCII text; an approved markup language format; or an approved non-ASCII format; and

(c) use the objects, interchange formats, document formats, and file name extensions below:

Format	Object	Interchange Format	Document Format	File Extension
Plain ASCII Text	ASCII_DOCUMENT	ASCII	TEXT	.ASC
HTML	HTML_DOCUMENT	ASCII	TEXT	.HTM or .HTML*
TeX	TEX_DOCUMENT	ASCII	TEXT	.TEX
LaTeX	LATEX_DOCUMENT	ASCII	TEXT	.TEX
Adobe PDF	PDF_DOCUMENT	BINARY	ADOBE PDF	.PDF
MS Word	WORD_DOCUMENT	BINARY	MICROSOFT WORD	.DOC
Rich Text	RTF_DOCUMENT	BINARY	RICH TEXT	.RTF
GIF	GIF_DOCUMENT	BINARY	GIF	.GIF
JPG	JPG_DOCUMENT	BINARY	JPG	.JPG
Encapsulated Postscript	EPS_DOCUMENT	BINARY	ENCAPSULATED POST SCRIPT	.EPS
Postscript	PS_DOCUMENT	BINARY	POSTSCRIPT	.PS
Tagged Image File Format	TIFF_DOCUMENT	BINARY	TIFF	.TIF or .TIFF*

* See chapter *File Specification and Naming* regarding extensions with more than three characters.

Example: A document defined as a DOCUMENT object appearing in the DOCUMENT directory might be the following:

1. MYDOC.ASC - required ASCII version
2. MYDOC.DOC - optional Microsoft Word version to retain all graphics
3. MYDOC001.TIF - optional scanned TIFF version of selected pages
4. MYDOC002.TIF - optional scanned TIFF version of other selected pages
5. MYDOC.LBL - PDS label defining DOCUMENT object(s) for these files

Optional versions of the document should have the same file name as the required ASCII version but with different extensions. Optional versions should be defined as DOCUMENT objects in the PDS label; the name of the required ASCII file should be indicated using the DESCRIPTION keyword.

9.2 Document Format Details

9.2.1 Plain, Unmarked ASCII Text

Line Delimiters / Line Length - PDS recommends that lines of plain ASCII text be limited to 78 or fewer characters. The line length recommendation is made to facilitate importation of text into environments which may reserve several characters for line numbering or other uses. Regardless of length, each line must be followed by the Carriage Return (ASCII decimal 13) and Line Feed (ASCII decimal 10) characters. The use of the Carriage Return and Line Feed characters ensures readability in the four environments commonly in use by planetary researchers. In the Macintosh and Unix environments simple utilities are available (Apple File Exchange and Translate, respectively) to add (if submitting data) or eliminate (if using data) the Line Feed or Carriage Return.

Page Break / Page Length - Paragraphs should be separated by one or more empty lines-ASCII blanks only (if used for padding) and the Carriage Return/Line Feed sequence. This facilitates simple conversion of text files into word processor formats. In order to organize text into pages, the Page Feed (ASCII decimal 12) is allowed. If this feature is used, page length should be limited to 60 lines of text and a Page Feed character should be inserted immediately after the END (Carriage Return/Line Feed) statement of any PDS labels which appear at the beginning of the file.

9.2.2 ASCII Text Containing Markup Language

Line Delimiters / Line Length - Because these files conform to specific markup language constructs, the recommended line length of 78 or fewer characters does not apply. However, each line must be delimited by the Carriage Return (ASCII decimal 13) and Line Feed (ASCII decimal 10) characters.

Page Break / Page Length - Because these files conform to specific markup language constructs and because the Page Feed (ASCII decimal 12) character may not function as such, the recommended page length limit does not apply.

Note: There are several markup languages which are not recommended because the markup is so extensive that it inhibits easy reading of the text portions of the file. There are also automatic generators or converters of files that create output in approved formats which are also difficult to read. Such files may be judged unsuitable for meeting the 'ASCII text' requirement.

The following subsections describe PDS standards applicable to HTML files.

9.2.2.1 HyperText Markup Language (HTML) Version

PDS archive products must adhere to Version 3.2 of the HTML language, a Standard Generalized Markup Language which conforms to International Standard ISO 8879. All files are subject to validation against the HTML 3.2 SGML Declaration and the HTML Document Type Definition.

Note: Constructs not defined in the HTML 3.2 standard (e.g., FRAME, STYLE, SCRIPT, and FONT FACE tags) are not allowed in PDS document files.

9.2.2.2 Location of Files

PDS strongly recommends that targets of all HTML links be present on the archive volume. In cases where external links are provided, the link should lead to supplementary information which is not essential to understanding or use of the archival data.

PDS recommends that all files which comprise an HTML document-or series of documents-be co-located in a single directory. However, locating ancillary files (e.g., images, common files) in subdirectories may be required under certain circumstances (e.g., to avoid conflicts in file names and to minimize replication of common files).

9.2.2.3 Character Set

Documentation files prepared to accompany the data set or data set collection must be validated in that the files can be copied or transmitted electronically, and can be read or printed by their target text processing program. Documentation files should be spell-checked prior to being submitted to PDS for validation.

9.2.2.4 Excluded / Discouraged HTML 3.2 Capabilities

Although the APPLET tag is advertised to be supported by all Java enabled browsers, not all applets execute on all browsers on all platforms. Further, some browsers require that the user explicitly enable use of Java applets before the applet will execute. Applets are permitted only when the information they convey is not essential to understanding or use of the archival data.

Use of the TAB character is permitted but strongly discouraged because of variations in implementation among browsers and resulting misalignments within documents.

Use of animated GIFs is discouraged.

9.2.3 Non-ASCII Formats

Where possible the version or ENCODING_TYPE should be specified for the format used. For example, a PostScript or Encapsulated PostScript file could be further documented by noting the generating version with the DESCRIPTION keyword. The label defining a GIF document might include the keyword-value pair ENCODING_TYPE = "GIF87A".

9.3 Examples

9.3.1 Simple Attached label (Plain ASCII Text)

The following label could be prepended to a plain ASCII text file which describes the content and format of Mars Pathfinder Imager Experiment Data Records. The aggregate file could be stored under the file name EDRSIS.TXT.

```
PDS_VERSION_ID          = PDS3
RECORD_TYPE             = STREAM
OBJECT                  = TEXT
 NOTE                   = "Mars Pathfinder Imager Experiment Data Record SIS"
 PUBLICATION_DATE       = 1998-06-30
END_OBJECT              = TEXT
END
```

9.3.2 Complex Detached Label (Two Document Versions)

If the data producer chose to provide the same document in both plain ASCII text and as a Microsoft Word document, the detached label would have the name EDRSIS.LBL and would be as follows:

```
PDS_VERSION_ID          = PDS3
RECORD_TYPE             = UNDEFINED
^ASCII_DOCUMENT         = "EDRSIS.ASC"
^WORD_DOCUMENT          = "EDRSIS.DOC"

OBJECT                  = ASCII_DOCUMENT
 DOCUMENT_NAME          = "Mars Pathfinder Imager Experiment Data Record"
 PUBLICATION_DATE       = 1998-06-30
 DOCUMENT_TOPIC_TYPE    = "DATA PRODUCT SIS"
 INTERCHANGE_FORMAT     = ASCII
 DOCUMENT_FORMAT             = TEXT
 DESCRIPTION                 = "This document contains a textual description
                                of the VICAR and PDS formatted Mars Pathfinder
                                IMP Experiment Data Records.  This is the ASCII
                                text version of the document required by PDS."
END_OBJECT                   = ASCII_DOCUMENT

OBJECT                       = WORD_DOCUMENT
 DOCUMENT_NAME               = "Mars Pathfinder Imager Experiment Data Record"
 PUBLICATION_DATE            = 1998-06-30
 DOCUMENT_TOPIC_TYPE         = "DATA PRODUCT SIS"
 INTERCHANGE_FORMAT          = BINARY
 DOCUMENT_FORMAT             = "MICROSOFT WORD"
 DESCRIPTION                 = "This document contains a textual description
                                 of the VICAR and PDS formatted Mars Pathfinder
                                 IMP Experiment Data Records.  The document was
                                 created using MicroSoft Word 6.0.1 for the Macintosh."
END_OBJECT                    = WORD_DOCUMENT
END
```

9.3.3 Complex Detached Label (Documents Plus Graphics)

The following label (EDRSIS.LBL) illustrates the use of an HTML document as the required ASCII document. The same document is also included as a PDF file, and four GIF images are included separately.

```
PDS_VERSION_ID               = PDS3
RECORD_TYPE                  = UNDEFINED
^HTML_DOCUMENT               = "EDRSIS.HTM"
^PDF_DOCUMENT                = "EDRSIS.PDF"
^GIF_DOCUMENT                = ("FIG1.GIF","FIG2.GIF","TAB1.GIF","TAB2.GIF")

OBJECT                       = HTML_DOCUMENT
 DOCUMENT_NAME               = "Mars Pathfinder Imager Experiment Data Record"
 PUBLICATION_DATE            = 1998-06-30
 DOCUMENT_TOPIC_TYPE         = "DATA PRODUCT SIS"
 INTERCHANGE_FORMAT          = ASCII
 DOCUMENT_FORMAT             = HTML
 DESCRIPTION                 = "This document contains a textual description
                                of the VICAR and PDS formatted Mars Pathfinder
                                IMP Experiment Data Records.  This is the ASCII
                                text version of the document required by PDS."
END_OBJECT                   = HTML_DOCUMENT

OBJECT                       = PDF_DOCUMENT
 DOCUMENT_NAME               = "Mars Pathfinder Imager Experiment Data Record"
 PUBLICATION_DATE            = 1998-06-30
 DOCUMENT_TOPIC_TYPE         = "DATA PRODUCT SIS"
 INTERCHANGE_FORMAT          = BINARY
 DOCUMENT_FORMAT             = "ADOBE PDF"
DESCRIPTION                  = "This document contains a textual description
                                of the VICAR and PDS formatted Mars Pathfinder
                                IMP Experiment Data Records.  This is the ASCII
                                text version of the document required by PDS."
END_OBJECT                   = PDF_DOCUMENT

OBJECT                       = GIF_DOCUMENT
 DOCUMENT_NAME               = "Mars Pathfinder Imager Experiment Data Record"
 PUBLICATION_DATE            = 1998-06-30
 DOCUMENT_TOPIC_TYPE         = "DATA PRODUCT SIS"
 FILES                  = 4
 ENCODING_TYPE          = "GIF89A"
 INTERCHANGE_FORMAT     = BINARY
 DOCUMENT_FORMAT        = GIF
DESCRIPTION             = "This document is a GIF89A representation of
                           two figures and two tables from the Mars
                           Pathfinder IMP Experiment Data Record SIS.
                           The figures and tables are included in the
                           PDF version of the document; only the tables
                           are included in the HTML version -- and then
                           in a somewhat different presentation than in
                           the original, printed version of the document.
                           The files use the naming convention FIGxx.GIF,
                           where xx is the figure number, and TABxx.GIF,
                           where xx is the table number, in the file."
END_OBJECT              = GIF_DOCUMENT
END
```

Chapter 10

File Specification and Naming

The File Specification and Naming Standard defines the PDS conventions for forming file specifications and file names. This standard is based on Level 1 and Level 2 of the international standard ISO 9660, "Information Processing - Volume and File Structure of CD-ROM for Information Interchange."

ISO 9660 Level 1 versus ISO 9660 Level 2

- PDS recommends that archive products comply with the ISO 9660 Level 1 specification. Specifically, PDS recommends that CD-ROM volumes which are anticipated to have a wide distribution use file identifiers consisting of a maximum of 8 characters for the file name and 3 characters for the extension to comply with the ISO 9660 Level 1 specification.

- PDS has adopted the ISO 9660 Level 2 specification for those archive products for which there are compelling reasons to relax the 8.3 filename restrictions.

10.1 File Specification Standards

A file specification consists of the following elements:

A complete directory path name	(as discussed in the *Directory Types and Naming* chapter of this document)
A file name having an extension	

The PDS has adopted the UNIX/POSIX forward slash operator (/) for use in path names. Directory path name formation is discussed further in the *Directory Types and Naming* chapter of this document.

The following is an example of a simple file specification. The file specification identifies the location of the file relative to the root of a volume, inlcuding the directory path name.

File Name:	TG15N122.IMG
File Specification:	TG15NXXX/TG15N1XX/TG15N12X/TG15N122.IMG

Do not use path or file names that correspond to operating system specific names, such as:

AUX COM1 CON LPT1 NUL PRN

10.1.1 ISO 9660 Level 1 Specification

A file name consists of a basename and an extension separated by a required FULL STOP (a.k.a. period) character (.). The total length of the file name shall not exceed 12 characters. The length of the base name shall not exceed 8 characters and the extension shall not exceed 3 characters. The file identifier must be suffixed with a version number consisting of a semicolon and an integer to comply with the ISO 9660 Level 1 specification. Both the base name and extension shall contain only the upper case alphanumeric character set (A- Z, 0-9), and underscore (_). These requirements are often referred to as the 8.3 (8 dot 3) file naming convention. These limitations exist primarily to accommodate older computer systems (e.g. IBM DOS-based PCs) that cannot handle longer file names. Since PDS archive volumes are designed to be read on many platforms, including PCs, these restrictions are necessary.

Preferred format: FILENAME (1..8 characters) "." EXTENSION (3 characters)

Allowable format: FILENAME (1..8 characters) "." EXTENSION (1..3 characters)

Actual format
on archive media: FILENAME (1..8 characters) "." EXTENSION (1..3 characters) ";1"

10.1.2 ISO 9660 Level 2 Specification

A file name consists of a basename and an extension separated by a required FULL STOP (a.k.a. period) character (.). The total length of the file name shall not exceed 31 characters. Both the base name and extension shall contain only the upper case alphanumeric character set (A- Z, 0-9), and underscore (_). These requirements are often referred to as the 27.3 (27 dot 3) file naming convention. The file identifier must be suffixed with a version number consisting of a semicolon and an integer to comply with the ISO 9660 Level 2 specification. It is strongly recommended that the PDS file naming recommendations for file extensions be followed (including the use of the standard 3 character extensions).

Preferred format: FILENAME (1..27 characters) "." EXTENSION (3 characters)

Allowable format: FILENAME (1..29 characters) "." EXTENSION (1..29 characters)

Actual format
on archive media: FILENAME (1..29 characters) "." EXTENSION (1..29 characters) ";1"

10.2 File Naming Standards

The following sections identify the PDS required and reserved file names and file extensions. Required and reserved file names and extensions provide consistency across PDS archive volumes, which is helpful to users. Also, software tools can make use of this predictability.

Required means that if a file contains a given type of information, it shall have the given name or extension. Reserved means that if a file has a given name or extension, it shall contain that type of information. For example, the volume object is contained in, and only in, the file named VOLDESC.CAT. It is a required file name. A file named TG15N122.IMG contains an image. File extensions should be used to identify the data type of a file. This is reflected in the required and reserved file extensions listed later in this chapter.

10.2.1 Required File Names

VOLDESC.CAT - This file name must be used for the file containing the volume object. This required file is placed in the ROOT directory of a volume.

*objectname.*CAT - This category of file name must be used for files containing a catalog object. These files, if present on a volume, must be placed in the CATALOG directory of a volume. The Software Inventory catalog object may also be placed in the SOFTWARE hierarchy under the appropriate DOC directory.

"*objectname*" is one of the commonly used catalog objects listed below. The form of the file name varies if one or more objects are included in the archive product. For example, if a volume contains a single data set, the data set object shall be contained in the file named DATASET.CAT. If the volume contains multiple data sets and the data set objects are contained in separate files, each file shall be named xxxxxxDS.CAT where "xxxxxx" is replaced with an acronym of up to six characters for the data set.

It is possible to include all of the catalog objects in a single file. However, PDS strongly discourages this approach. One of the advantages to supplying individual files is that each catalog object can be 'plugged into' other archive products. If a single file is used to contain all catalog objects, it must be named CATALOG.CAT. The pointer expression becomes:

```
^CATALOG = "CATALOG.CAT"
```

If catalog objects are organized in separate files or sets of files, pointer expressions shall be constructed according to the following table. Under "File Name", the first line shows the file name to be used if a single catalog file is present on the volume for the particular type of catalog object named. The second shows the syntax and file name convention to be followed if multiple catalog files are present for the named object.

Catalog Pointer Name		**File Name**
^DATA_SET_CATALOG	=	"DATASET.CAT"
	=	{"xxxxxxDS.CAT","yyyyyyDS.CAT"}
^DATA_SET_COLLECTION_CATALOG	=	"DSCOLL.CAT"
	=	{"xxxxxDSC.CAT","yyyyyDSC.CAT"}
^DATA_SET_MAP_PROJECTION_CATALOG	=	"DSMAP.CAT"
	=	{"xxxDSMAP.CAT","yyyDSMAP.CAT"}
^INSTRUMENT_CATALOG	=	"INST.CAT"
	=	{"xxxxINST.CAT","yyyyINST.CAT"}
^INSTRUMENT_HOST_CATALOG	=	"INSTHOST.CAT"
	=	{"xxxxHOST.CAT","yyyyHOST.CAT"}

```
^MISSION_CATALOG                    =   "MISSION.CAT"
                                    =   {"xxxxxMSN.CAT","yyyyyMSN.CAT"}
^PERSONNEL_CATALOG                  =   "PERSON.CAT"
                                    =   {"xxxxPERS.CAT,"yyyyPERS.CAT"}
^REFERENCE_CATALOG                  =   "REF.CAT"
                                    =   {"xxxxxREF.CAT","yyyyyREF.CAT"}
^SOFTWARE_INVENTORY_CATALOG         =   "SWINV.CAT"
                                    =   {"xxxSWINV.CAT", "yyySWINV.CAT"}
^TARGET_CATALOG                     =   "TARGET.CAT"
                                    =   {"xxxTGT.CAT", "yyyTGT.CAT"}
```

AAREADME.TXT- This file name must be used for the file that contains a terse description of the volume contents. This required file is placed in the ROOT directory of a volume.

ERRATA.TXT- This file name is used for a file used to provide comments as well as to report errors. Cumulative comments for a volume set are kept in this file (although cumulative comments are optional for a volume set). This optional file is placed in the ROOT directory of a volume.

VOLINFO.TXT - This file name must be used for the file containing detailed information necessary to interpret the data set(s) contained on the volume. When present, this file is placed in the DOCUMENT directory of a volume. The VOLINFO.TXT file is referenced in the catalog object as ^DESCRIPTION = "VOLINFO.TXT".

NOTE: PDS strongly discourages the use of the VOLINFO.TXT file. This file can be provided as an alternate for individual catalog objects, but this approach negates the ability to re-use each catalog object in other archive products. PDS requires that either the VOLINFO.TXT file, or the *objectname*.CAT files described above, be present on the volume.

The following xxINFO.TXT files are required to appear in the non-data subdirectories that appear on the volume:

Sub-directory	**File**
BROWSE	BROWINFO.TXT
CALIB	CALINFO.TXT
CATALOG	CATINFO.TXT
DOCUMENT	DOCINFO.TXT
EXTRAS	EXTRINFO.TXT
GAZETTER	GAZINFO.TXT
GEOMETRY	GEOMINFO.TXT
INDEX	INDXINFO.TXT
LABEL	LABINFO.TXT
SOFTWARE	SOFTINFO.TXT

The following xxINFO.TXT files are recommended in appropriate SOFTWARE subdirectories:

Sub-directory	File
SOFTWARE/PC	PCINFO.TXT
SOFTWARE/MAC	MACINFO.TXT
SOFTWARE/SUN	SUNINFO.TXT
SOFTWARE/SGI	SGIINFO.TXT

The following file names should be used for INDEX files:

	Single Index	Multi-Index
Single Volume	INDEX.TAB	axxINDEX.TAB
Multi-volume set	INDEX.TAB and CUMINDEX.TAB	axxINDEX.TAB and axxCMIDX.TAB

10.2.2 Reserved File Names

VOLDESC. SFD - for use with a file containing an SFDU Reference Class object for an archive volume. Note that this file is identified here for backward compatability with previous versions of the PDS standards and is not to be used in current archive products.

10.2.3 Required File Extensions

.CAT - for use with a file containing a catalog object

.FMT - for use with an include file containing structural information (meta data) describing a data object

.LBL - for use with a file containing a detached PDS label for any class of data object. Note that a file containing a detached label should have the same base name as its associated data file, but the extension .LBL

.TXT - for use with a file described by the TEXT data object

.ASC - for use with a file containing a document in ASCII text format described by a label containing a DOCUMENT object definition.

10.2.4 Reserved File Extensions

Extension	Description (for use with a file containing)
DAT	Binary files (other than images)
TAB	Table data (Note: this extension is also used for table data in ASCII form described by a detached PDS label)
IMG	Image data
IBG	Browse image data
IMQ	Image data that have been compressed
HTM	HTML document
TEX	TeX or LaTeX document
PDF	Adobe PDF document
DOC	Microsoft Word document
RTF	Rich Text document
PS	Postscript
EPS	Encapsulated Postscript
GIF	GIF image
JPG	JPEG image
QUB	Spectral (or other) image qubes
XSP	SPICE Transfer format SPK (ephemeris) files
BSP	SPICE Binary format SPK (ephemeris) files
XC	SPICE Transfer format CK (pointing) files
BC	SPICE Binary format CK (pointing) files
TI	SPICE Text IK (instrument parameters) files
TLS	SPICE Leapseconds kernel files
TPC	SPICE Physical and cartographic constants kernel files
TSC	SPICE Spacecraft clock coefficients kernel files
XES	SPICE E-kernel files.
EXE	Application or Executable
MAK	Makefile for compiling / linking application or executable
OBJ	Object file
DLL	Dynamic Link Library
LIB	Library of object files
ZIP	Zip-compressed files within PDS.

NOTE: Additional file extensions are reserved for use for document files only and are described in the *Documentations* chapter in this document.

10.3 File Naming Guidelines

In cases where file names will contain an identification value constructed from the time tag or data object identifier, the following forms are suggested (but not required):

Pnnnnnnn.EXT

where P is one of the following:
C - The following value is a clock count value (C3345678.IMG)
T - The following value is a time value (T870315.TAB)
F - The following value is a FrameID or an ImageID (F242AO3.IMG)
N - The following value is a numeric file identification number (N003.TAB).

Chapter 11

Media Formats for Data Submission and Archive

This standard identifies the physical media formats to be used for data submission or delivery to the PDS or its Science Nodes. It is expected that flight projects will deliver all standard digital products on magnetic or optical media. Electronic delivery of modest volumes of special science data products may be negotiated with the Science Nodes.

During archive planning, the data producer and PDS will determine the medium (or media) to use for data submission and archive. This standard lists the media that are most commonly used. Delivery of data on media other than those listed here can be negotiated with PDS on a case-by-case basis.

The use of 12-inch Write Once Read Many (WORM) disk, 8-mm EXABYTE tape or 4-mm DAT tape is NOT recommended for archival products. WORM disks are not transportable between various vendor hardware. Helical scan tape (8-mm or 4-mm) is prone to catastrophic read errors.

For archival products only media that conform to International Standards Organization (ISO) standards for physical and logical recording formats should be used.

1. The preferred data delivery medium is the compact disc, either CD-ROM or CD-WO (recordable) disc, in ISO-9660 format, using Interchange Level 1.

2. Standard computer compatible tape (CCT) on 12-inch reels recorded in ANSI format (equivalent to VAX 'COPY' format) is acceptable.

3. ISO compatible 5 1/4-inch WORM or Magneto Optical disk is acceptable.

4. IBM 3480-compatible tape cartridges are acceptable.

11.1 CD-ROM Recommendations

11.1.1 Use of Extended Attribute Records (XARs)

The use of Extended Attribute Records (XARs) on CD-ROMs shall be at the discretion of the data producer, based on the anticipated use of the CD-ROMs. If the CD-ROMs will be widely used on VMS platforms with software which expects certain record formats, then XARs should be provided. If the CD-ROMs will be used on mixed platforms and there is no existing software on the VMS platform which accesses the data files, XARs need not be included. This issue should be discussed during the Peer Review or Data Delivery Review for any CD-ROM product.

See the *Record Formats* chapter of this document for additional requirements on CD-ROMs that have XARs.

Software developed by PDS for use on VMS platforms should not expect record attributes to be specified on all CD-ROM data files, and should allow processing of files which do not have XAR records. Preferably, they should extract information about the record attributes from the PDS labels, not from the operating system.

11.1.2 Premastering Recommendation

PDS recommends that CD-ROMs be premastered using a single-session, single-track format. Other formats have been found to be incompatible with some readers.

11.1.3 Packaging Software files on a CD-ROM

If the archive is being premastered such that it will be supported on all platforms and it includes software for the MAC and SUN, then the following applies:

1. MAC Software

If the archive includes software for the MAC, the MAC files must be prepared in a particular format. This is because other platforms can't recognize the resource and data fork files that come with MAC applications. This has been done with the NIHIMAGE software on the Magellan GxDR and the Clementine EDR CD-ROMs. There is a MAC utility, called STUFFIT, that is used to prepare the files; i.e. compress and BINHEX the MAC files. The users will also need this utility in order to use the software (they will need to unBINHEX and decompress the file). This should be described in a text file included on the CD-ROM (in the appropriate SOFTWARE/DOC subdirectory).

```
----------------------------------------------------------------------
Example of text documenting HQX files
----------------------------------------------------------------------

                Macintosh Software

This directory contains software which can be used to display the GXDR
images on a Macintosh II computer with an 8-bit color display.

NOTE:  Because of the way this CD-ROM was produced, it was not
possible to record this display program as a Macintosh executable
file.  Anyone who is unfamiliar with the Macintosh STUFFIT utility
should contact the PDS operator, 818-306-6026, SPAN address
JPLPDS::PDS_OPERATOR, INTERNET address PDS_OPERATOR@JPLPDS.JPL.NASA.GOV

The file IMAGE.HQX contains the NIH Image program, along with several
ancillary files and documentation in Microsoft WORD format.  It was
written by Wayne Rasband of the National Institutes of Health. The
program can be used to display any of the image files on the GXDR
CD-ROM disks.

The Image executable and manual are stored in BINHEX format, and the
utility STUFFIT or UNSTUFFIT must be used to:  1) decode the BINHEX
```

file IMAGE.HQX into IMAGE.SIT, using the 'DECODE BINHEX FILE...' option in the Other menu; and 2) use 'OPEN ARCHIVE' from the File menu to extract Image 1.40 from the STUFFIT archive file. There are also several other files in the archive file which should be unstuffed and kept together in the same folder as the Image executable is stored.

The STUFFIT software is distributed as shareware. STUFFIT, Version 1.5.1, is available by contacting:

Raymond Lau MacNET:RayLau Usenet:raylau@dasys1.UUCP
100-04 70 Ave. GEnie:RayLau
Forest Hills, N.Y. 11375-5133 CIS:76174,2617
United States of America. Delphi:RaymondLau

Alternatively, STUFFIT CLASSIC, Version 1.6, is available by contacting:

Aladdin Systems, Inc.
Deer Park Center
Suite 23A-171
Aptos, CA 95003
United States of America

2. SUN Software - preserving the SUN filesystem (e.g. filenames)

The ISO standard is all files and directories are uppercase, so when a disc is premastered as an ISO CD, this is automatically done by the premastering software. We know from experience that some CD readers connected to SUNs can show files/directories as uppercase instead of lowercase. This can cause problems when the user copies the files over and tries to do a build if the software filename should be lowercase.

There are two options on how to preserve the SUN filesystem (other than not doing anything and just documenting it). The first option was used for Clementine.

The options are:
a. Build tar/compressed/encoded files for the SUN executables and source files. This is analogous to what is done for the MAC with the HQX files. This way the actual software filenames will be retained as they should be for the SUN when the user copies over the files and decodes/ uncompresses/detars them. This should be documented.

b. YoungMinds provides something to deal with this very problem. A translation table can be created (called YMTRANS.TBL) to provide a mapping of the filename on the CD to what it should be on the SUN UNIX. If the premastering is on a PC, this can't be done automatically because the files have already been moved to a PC. However, it is only an ASCII table with a simple format so it can be created manually. There would have to be a translation table in every SUNOS subdirectory (/BIN, /SOURCE, /DOC) and its contents should only be of the files that appear in the subdirectory in which it exists. Software must be provided on the CD (provided by YoungMinds) for the user to copy the files. This software uses the translation tables. This would also have to be documented. As an alternative to the Young Minds solution, one could supply a custom script with the CD that will perform the proper case translations.

Chapter 12

Object Description Language Specification and Usage

The following provides a complete specification for the Object Description Language (ODL), the language used to encode data labels for the Planetary Data System (PDS) and other NASA data systems. This standard contains a formal definition of the grammar of the ODL and describes the semantics of the language. PDS specific implementation notes and standards are referenced in separate sections of this chapter.

12.1 About the ODL Specification

This standard describes Version 2.1 of the ODL. Version 2.1 of ODL supersedes Versions 0 and 1 of the language which were used previously by the PDS and other groups. For the most part, ODL Version 2.1 is upwardly compatible with previous versions of ODL. There are, however, some features found in ODL Versions 0 and 1 that have been removed from or changed within Version 2. The differences between ODL versions are described in Section 12.7. The following is a sample data label written in ODL that describes a file and its contents:

```
/* File Format and Length */
        RECORD_TYPE             = FIXED_LENGTH
        RECORD_BYTES            = 800
        FILE_RECORDS            = 860
/* pointer to First Record of Major Objects in File */
        ^IMAGE                  = 40
        ^IMAGE_HISTOGRAM        = 840
        ^ANCILLARY_TABLE        = 842
/* Image Description */
        SPACECRAFT_NAME         = VOYAGER_2
        TARGET_NAME             = IO
        IMAGE_ID                = "0514J2-00"'
        IMAGE_TIME              = 1979-07-08T05:19:11
        INSTRUMENT_NAME         = NARROW_ANGLE_CAMERA
        EXPOSURE_DURATION       = 1.9200 <SECONDS>
        NOTE                    = “Routine multispectral longitude
                                    coverage,1 of 7 frames”
/* Description of the Objects Contained in the File */
        OBJECT                  = IMAGE
         LINES                  = 800
         LINE_SAMPLES           = 800
         SAMPLE_TYPE            = UNSIGNED_INTEGER
         SAMPLE_BITS            = 8
        END_OBJECT              = IMAGE

        OBJECT                  = IMAGE_HISTOGRAM
        ITEMS                   = 25
        ITEM_TYPE               =INTEGER
        ITEM_BITS               = 32
        END_OBJECT              = IMAGE_HISTOGRAM
```

```
OBJECT                    = ANCILLARY_TABLE
 ^STRUCTURE               = “TABLE. FMT”
 END_OBJECT               = ANCILLARY_TABLE
END
```

12.1.1 Implementing ODL

Notes to implementers of software to read and write ODL-encoded data descriptions appear throughout the following sections. These notes deal with issues that are beyond language syntax and semantics but that are addressed to assure that software for reading and writing ODL will be uniform. The PDS, which is the major user of ODL-encoded data labels, has levied additional implementation requirements for software used within the PDS and these requirements are discussed below where appropriate.

12.1.1.1 Language Subsets

Implementers are allowed to develop software to read or write subsets of the ODL. Specifically, software developers may:

- Eliminate support for the GROUP statement (see Section 12.4.5.2 for additional information)
- Not support pointer statement
- Not support certain types of data values

For every syntactic element supported by an implementation, the corresponding semantics must be fully supported, as spelled out in this document. Software developers should be careful to assure that language features will not be needed for their particular applications before eliminating them. Documentation on label reading/writing software should clearly indicate whether or not the software supports the entire ODL and if the software does not support the full ODL specification, the documentation should clearly spell out the subset that is allowed.

12.1.1.2 Language Supersets

Software for writing ODL must not provide or allow lexical or syntactic elements over and above those described below. With the exception of the PVL-specific extensions below, software for reading ODL must not provide or allow any extensions to the language.

12.1.1.3 PDS Implementation of PVL-Specific Extensions

PDS implementation of software for reading ODL may, in some cases, provide handling of lexical elements which are included in the CCSDS specification of the Parameter Value Language (PVL). PVL is a superset of ODL. Extensions which may be handled by such software include:

- BEGIN_OBJECT as a synonym for the reserved word OBJECT.
- BEGIN_GROUP as a synonym for the reserved word GROUP.
- Use of the semicolon (;) as a statement terminator.

These lexical elements will not be supported by software which writes ODL.Therefore, they must be removed (in the case of semicolons) or replaced (in the case of the BEGIN_OBJECT and BEGIN_GROUP synonyms) upon output.

12.1.2 Notation

The formal description of the ODL grammar is given in a Backus-Naur format (BNF) notation. Language elements are defined using rules of the following form:

defined_element ::= definition

where the definition is composed from the following components:

1. Lower case words, some containing underscores, are used to denote syntactic categories. For example:

units_expression

Whenever the name of a syntactic category is used outside of the formal BNF specification, spaces take the place of underscores (for example, units expression).

2. Boldface type is used to denote reserved identifiers. For example:

object

Special characters used as syntactic elements also appear in boldface type.

3. Square brackets enclose optional elements. The elements within the bracket may occur once at most.

4. Square brackets followed immediately by an asterisk or plus sign specify repeated elements. If the asterisk follows the brackets, the elements in the brackets may appear zero, one, or more times. If a plus sign follows the brackets, the elements in the brackets must appear at least once. The repetitions occur from left to right.

5. A vertical bar separates alternative elements.

6. If the name of any syntactic category starts with an italicized part, it is equivalent to the category name without the italicized part. The italicized part is

intended to convey some semantic information. For example, both *object*_identifier and *units*_identifier are equivalent to identifier; *object*_identifier is used in places where the name of an object is required and *units*_identifier is used where the name of some unit of measurement is expected.

12.2 Character Set

The character set of ODL is the International Standards Organization's ISO 646 character set. The U.S. version of the ISO 646 character set is ASCII and the ASCII graphical symbols are used throughout this document. In other countries certain symbols have a different graphical representation.

The ODL character set is partitioned into letters, digits, special characters, spacing characters, format effectors and other characters:

```
character :: =   letter | digit | special_character |
                 spacing_character | format_effector |
                 other_character
```

The letters are the uppercase letters A - Z and the lowercase letters a - z. The ODL is case-insensitive, meaning that lower case letters are treated as identical to their upper-case equivalent. Thus the following identifiers are equivalent:

- IMAGE_NUMBER
- Image_Number
- image_number

An exception to the case rule is when lowercase letters appear as part of text strings. For example, the text String “abc” is not the same as the string “ABC”.

The digits are 0, 1,2,3,4, 5, 6,7,8,9.
The following special characters are used in the ODL:

Symbol	**Name**	**Usage**
=	Equals	The equals sign equates an attribute or pointer to a value.
{ }	Braces	Braces enclose an unordered set of values.
( )	parentheses	parentheses enclose an ordered sequence of values.
+	Plus	The plus sign indicates a positive numeric value.
-	Minus	The minus sign indicates a negative numeric value.
< >	Angle brackets	Angle brackets enclose a units expression associated with a numeric value.
.	Period	The period is the decimal place in real numbers.
"	Quotation Marks	Quotation marks denote the beginning and end of a text string value.
’	Apostrophe	Apostrophes mark the beginning and end of a literal value.

_	Underscore	The underscore separates words within an identifier.
,	Comma	The comma separates the individual values in a set or sequence.
/	Slant	The slant character indicates division in units exp ressions. The slant is also part of the comment delimiter.
*	Asterisk	The asterisk indicates multiplication in units expressions. Two asterisks in
a		
		row indicate exponentiation in units expressions. The asterisk is also part of the comment delimiter.
:	Colon	The colon separates hours, minutes and seconds within a time value.
#	Sharp	The sharp delimits the digits in an integer number value expressed in based notation.
&	Ampersand	The ampersand denotes continuation of a statement onto another line.
^	Circumflex	The circumflex indicates that a value is to be interpreted as a pointer.

Two characters, called the spacing characters, separate lexical elements of the language and can be used to format characters on a line:

Space
Horizontal Tabulation

The following ISO characters are format effectors, used to separate ODL encoded statements into lines:

Carriage Return
Line Feed
Form Feed
Vertical Tabulation

The spacing characters and format effectors are discussed further in section 12.4.1 below. There are other characters in the ISO 646 character set that are not required to write ODL statements and labels. These characters may, however, appear within text strings and quoted symbolic literals:

$ % ; ? @ [ ]’ | ˜

The category of other characters also includes the ASCII control characters except for horizontal tabulation, carriage return, line feed, form feed and vertical tabulation (e.g., the control characters that serve as spacing characters or format effectors). As with the printing characters in this category, the control characters in this category can appear within a text String or symbolic literal. The handling of control characters within text strings and symbolic literals is discussed in Section 12.3.5 below.

12.3 Lexical Elements

This section describes the lexical elements of the ODL. Lexical elements are the basic building blocks of the ODL and statements in the language are composed by stringing lexical elements together according to the grammatic rules presented in Section 12.4. The lexical elements of the ODL are:

- Numbers
- Dates and Times
- Strings
- Identifiers
- Special symbols used for operators, etc.

Lexical elements are technically only strings of characters and a lexical element has no meaning in and of itself: the meaning depends upon the syntactic role played by the element and the corresponding semantics. Therefore rules for determining the meaning of lexical elements (for example, the rules that govern the range of numeric values) are found in the sections on language syntax - sections 12.4 and 12.5 below - rather than in the current section. There is no limit on the length of any lexical element. However, software for reading and writing ODL may impose limitations on the length of text strings, symbol strings and identifiers. It is recommended that at least 32 characters be allowed for symbol strings and identifiers and at least 400 characters for text strings.

12.3.1 Numbers

The ODL can represent both integer numbers and real numbers. Integer numbers are usually represented in decimal notation (like 123), but the ODL also provides for integer values in other number systems (for example, 2#1111011# is the binary representation of the decimal integer number 123). Real numbers can be represented in simple decimal notation (like 123.4) or in a scientific notation that includes a base 10 exponent (for example, 1.234E2).

12.3.1.1 Integer Numbers In Decimal Notation

An integer number in decimal notation consists of a string of digits optionally preceded by a number sign. Unsigned integer numbers are assumed to be positive.

```
integer :: = [sign] unsigned_integer
unsigned_integer :: = [digit] +
sign ::=  + | -
```

<u>Examples of Decimal Integers</u>

```
   0
 123
+440
```

-150000

12.3.1.2 Integer Numbers In Based Notation

An integer number in based notation specifies the number base explicitly. The number base must be in the range 2 to 16, which allows for representations in the most popular number bases, including binary (base 2), octal (base 8) and hexadecimal (base 16). In general, for a number base X the digits 0 to X-1 are used. For example, in octal the digits 0 to 7 are allowed. If X is greater than 10, then the letters A, B, C, D, E, F (or their lower case counterparts) are used as needed for the additional digits.

A based integer may optionally include a number sign. An unsigned based integer number is assumed to be positive.

```
based_integer :: = radix # [sign] [extended_digit] + #
extended_digit :: = digit | letter
radix :: = unsigned_integer
```

Examples of Based Integers

```
2#1001011#
8#113#
10#75#
16#4B#
16#+4B#
16#-4B#
```

All but the last example above are equivalent to the decimal integer number 75. The final example is the hexadecimal representation of -75 decimal.

12.3.1.3 Real Numbers

Real numbers may be represented in a decimal notation (like 123.4) or in a scientific notation with a base 10 exponent specified (like 1.234E3). A real number may optionally include a number sign. Unsigned real numbers are assumed to be positive.

```
real :: = [sign] unscaled_real  | [sign] scaled_real
unscaled_real :: = unsigned_integer. [unsigned_integer] | .unsigned_integer
scaled_real :: = unscaled_real exponent
exponent :: = E integer | e integer
```

Note that the letter E in the exponent of a real number may appear in either upper or lower case.

Examples of Real Numbers

0.0

123.
+1234.56
-.9981
-1.E-3
31459e1

12.3.2 Dates and Times

The ODL has lexical elements to represent dates and times. The formats for dates and times are a subset of the formats defined by the International Standards Organization Draft Standard ISO/DIS 8601.

(For information regarding PDS specific use of dates and times, see the *Date/Time* chapter in this document.)

12.3.2.1 Date and Time Values

Date and time scalar values represent a date, or a time, or a combination of date and time:

date_time_value :: = date | time | date_time

The following rules apply to date values:

- The year must be in Anno Domini format (i.e., 1990). PDS requires a 4-digit year format be used (i.e., 2000).
- The month must be a number between 1 and 12.
- The day of month must be a number in the range 1 to 31, as appropriate for the particular month and year.
- The day-of-year must be in the range 1 to 365, or 366 in a leap year.

The following rules apply to time values:

- Hours must be in the range 0 to 23.
- Minutes must be in the range 0 to 59.
- Seconds, if specified, must be greater than or equal to 0 and less than 60.

The following rules apply to zone offsets within zoned time values:

- Hours must be in the range -12 to + 12 (the sign is mandatory).

- Minutes, if specified, must be in the range 0 to 59.

12.3.2.2 Implementation of Dates and Times

All ODL reading/writing software shall be able to handle any date within the 20th and 21st centuries.

Software for writing ODL shall always output full four-digit year numbers so that the labels will be valid into the next century.

Times in ODL may be specified with unlimited precision (for example, to nanoseconds). The actual precision with which times can be represented by label reading/writing software is determined by the software implementers, based upon limitations of the hardware on which the software is implemented. Developers of label reading/writing software should document the precision to which times can be represented.

Software for writing ODL shall not output local time values, since a label may be read in a time zone other than where it was written. Use either the UTC or zoned time format instead.

12.3.2.3 PDS Implementation of Dates and Times

PDS software for reading ODL labels shall interpret local times to be equivalent to UTC times. Upon output, a Z will be appended to local times.

12.3.2.4 Dates

Dates can be represented in two formats: as year and day-of-year; and as year, month and day of month.

date	:: = year_doy | year_month_day
year_doy	:: = year **-** doy
year_month_day	:: = year **-** month **-** day
year	:: = unsigned_integer
month	:: = unsigned_integer
day	:: = unsigned_integer
doy	:: = unsigned_integer

Examples of Dates

1990-07-04
1990-158
2001-001

12.3.2.5 Times

Times are represented as hours, minutes and optionally seconds using a 24-hour clock. Times may be specified in Universal Time Coordinated (UTC) by following the time with the letter Z (for Zulu, a common designator for Greenwich Mean Time). Alternately, the time can be referenced to any time zone by following the time with a number that specifies the offset from UTC. Most time zones are an integral number of hours from Greenwich, but some are different by some non-integral time, and both can be represented in the ODL. A time that is not followed by either the Zulu indicator or a time zone offset is assumed to be a local time.

```
time          :: = local_time | utc_time |  zoned_time
local time    :: = hour_min_sec
utc_time      :: = hour_min_sec Z
zoned_time    :: = hour_min_sec zone_offset
hour_min_sec  :: = hour: minute [:second]
zone_offset   :: = sign hour [: minute]
hour          :: = unsigned_integer
minute        :: = unsigned_integer
second        :: = unsigned_integer | unscaled_real
```

Note that either an integral or a fractional number of seconds can be specified in a time.

<u>Examples of Times</u>

```
12:00
15:24:12Z
01:10:39.4575+07  (time offset of 7 hours from UTC)
```

12.3.2.5.1 Combining Date and Time

A date and time can be specified together using the format below. Either of the two date formats can be combined with any time format - UTC, zoned or local.

date_time::=date **T** time

The letter T separating the date from the time can be specified in either upper or lower case. Note that because this is a lexical element that spaces may not appear within a date, within a time or before or after the letter T.

<u>Examples of Date/Times</u>

```
1990-07-04T12:00
1990-158T15:24:12Z
2001-001T01:10:39.457591+7
```

12.3.3 Strings

There are two kinds of string lexical elements in ODL: text strings and symbol strings.

12.3.3.1 Text Strings

Text strings are used to hold arbitrary strings of characters.

quoted_text ::= "[character]*"

The empty string -- a quoted text string with no characters within the delimiters -- is allowed.

A quoted text string may not contain the quotation mark, which is reserved to be the text string delimiter. A quoted text string may contain format effectors, hence it may span multiple lines in a label: the lexical element begins with the opening quotation mark and extends to the closing quotation mark, even if the closing mark is on a following line. The rules for interpreting the characters within a text string, including format effectors, are given in the section on string values in Section 12.5.

12.3.3.2 Symbol Strings

Symbol strings are sequences of characters used to represent symbolic values. For example, an image ID may be a symbol string like ’J123-U2A’, or a camera filter might be a symbol string like ’UV1.’

quoted_symbol ::= **’**[character]+**’**

A symbol string may not contain any of the following characters:

- The apostrophe character, which is reserved to be the symbol string delimiter
- Format effectors, which means that a symbol string must fit on a single line
- Control characters

12.3.4 Identifiers

Identifiers are used as the names of objects, attributes and units of measurement. They can also appear as the value of a symbolic literal.

Identifiers are composed of letters, digits, and underscores. Underscores are used to separate "words" in an identifier. The first character of an identifier must be a letter. The last character cannot be an underscore.

identifier : : = letter [letter | digit | **_**letter | **_**digit]*

Because ODL is not case sensitive, lower case characters in an identifier can be converted to their upper case equivalent upon input to simplify comparisons and parsing.

Examples of Identifiers

```
VOYAGER
VOYAGER_2
BLUE_FILTER
USA_NASA_PDS_1_0007
SHOT_1_RANGE_TO_SURFACE
```

12.3.4.1 Reserved Identifiers

A few identifiers have special significance in ODL statements and they are therefore reserved and cannot be used for any other purpose (for example, as the name of an object or an attribute):

end	end_group	end_object
group	object	begin_object

12.3.5 Special Characters

The ODL is a simple language and it is usually clear where one lexical element ends and another begins. Spacing characters of format effectors may appear before a lexical element, between any pair of lexical elements, or after a lexical element without changing the meaning of a statement.

As can be seen in the sections above, many lexical elements incorporate special characters. Examples are the decimal point in real numbers and the quotation marks that delimit a text string. Some special characters are lexical elements in their own right. These so-called delimiters appear within the syntax descriptions in the following section. The following single characters are delimiters unless they appear within one of the lexical elements described above or within a text or symbol string.

=	The equals sign is the assignment operator.
,	The comma separates the elements of an array or a set.
*	The asterisk serves as the multiplication operator in units expressions.
/	The slant serves as the division operator within units expressions.
^	The circumflex denotes a pointer to an object.
<>	The angle brackets enclose units expressions.
()	The parentheses enclose the elements of a sequence.
{ }	The braces enclose the elements of a set.

The following two-character sequence is a lexical element.

**	Two adjacent asterisks are the exponentiation sign within units expressions.

12.4 Statements

An ODL-encoded label is made up of a sequence of zero, one, or more statements followed by the reserve identifier **end**.

```
label ::= [statement]*

          end
```

The body of a label is built from four types of statements:

```
statement :: = attribute_assignment_statement |
pointer_statement |
object_statement |
group_statement
```

Each of the four types of statements is discussed below.

12.4.1 Lines and Records

Labels are also typically composed of lines, where each line is a string of characters terminated by a format effector or a string of adjacent format effectors. The following recommendations are given for how software that writes ODL should format a label into lines:

- There should be at most one statement on a line, although a statement may be more than a single line in length. As noted in Section 12.3.5 above, format effectors may appear before, after or between the lexical elements of a statement without changing the meaning of the statement. For example, the following statements are identical in meaning:
-

```
FILTER_NAME              = {RED, GREEN, BLUE}

FILTER_NAME              = {RED,
                             GREEN,
                             BLUE}
```

- Each line should terminate with a carriage return character followed immediately by a line feed character. This sequence is an end-of-line signal for most computer operating systems and text editors.
- The character immediately following the **end** statement must be either an optional spacing character or format effector, such as a space, line feed, carriage return, etc.

A line may include a comment. A comment begins with the two characters /* and ends with the two characters */. A comment may contain any character in the ODL character set except format effectors, which are reserved to mark the end of line (i.e., comments may not be more than one line long). Comments are ignored when parsing an ODL label. The comment delimiters (/* and */) may appear within a text string, but in this case, they do not represent a comment. They are simply part of the text string. For example, the following is not a correct use of a comment:

```
NOTE = “All good men come to the          /* Example of incorrect comment*/
         aid of their party"
```

Any characters on a line following a comment are ignored.

In some computer systems files are divided into records. Software for writing and reading ODL-encoded labels in record-oriented files should adhere to the following rules:

- A line of an ODL-encoded label should not cross a record boundary. Each line should be totally contained within a single record. Any space left over at the end of a record after the last line in the record should be set to all space characters.
- The remainder of the record that contains the end statement shall be ignored and the data portion of the file shall be assumed to begin with the next record in sequence.

12.4.2 Attribute Assignment Statement

The attribute assignment statement is the most common type of statement in ODL and is used to specify the value for an attribute of an object. The value may be a single scalar value, an ordered sequence of values, or an unordered set of values.

assignment_statement ::= *attribute*_identifier = value

The syntax and semantics of values are given in Section 12.5.

Examples of Assignments Statements

```
RECORD_BYTES              = 800
TARGET_NAME               = JUPITER
SOLAR_LATITUDE            = (0.25 <DEG>, 3.00 <DEG>)
FILTER_NAME               = {RED,
                             GREEN,
                             BLUE}
```

12.4.3 Pointer Statement

The pointer statement indicates the location of an object.

pointer_statement :: = ^*object*_identifier = value

As with the attribute assignment statement, the value may be a scalar value, an ordered sequence of values, or an unordered set of values.

A common use of pointer statements is to reference a file containing an auxiliary label. For example:

```
                    ^STRUCTURE = “TABLE.FMT"
```

is a pointer statement that points to a file name TABLE.FMT that contains a description of the structure of the ancillary table from our sample label. Another use of the pointer statement is to indicate the position of an object within another object. This is often used to indicate the position of major objects within a file. The following examples are from our sample label:

```
          ^ IMAGE                  = 40
          ^IMAGE_HISTOGRAM         = 840
          ^ANCILLARY_TABLE         = 842
```

The first pointer statement above indicates that the image is located starting at the 40th record from the beginning of the file. If an integer value is used to indicate the relative position of an object, the units of measurement of position are determined by the nature of the object. For files, the default unit of measurement is records. Alternatively, a units expression can be specified for the integer value to indicate explicitly the units of measurement for the position. For example, the pointer

```
          ^IMAGE          = 10200 <BYTES>
```

indicates that the image starts 10,200 bytes from the beginning of the file.

The object pointers above reference locations in the same files as the label. Pointers may also reference either byte or record locations in data files which are detached, or separate, from the label file:

```
          ^IMAGE          = ("IMAGE.DAT", 10)
          ^HEADER         = ("IMAGE.DAT", 512 <BYTES>)
```

12.4.4 OBJECT Statement

The OBJECT statement contains the description of an object. The description typically consists of a set of attribute assignment statements to establish the values of the object's attributes. If an object is itself composed of other objects, then OBJECT statements for the component objects may be nested within the object's description. There is no limit to the depth to which OBJECT statements can be nested.

The format of the OBJECT statement is:

object_statement :: = **object** = *object*_identifier
[statement]*
end_object [= *object*_identifier]

The object identifier gives a name to the particular object being described. For example, in a file containing images of several planets, the image object descriptions might be named VENUS_IMAGE, JUPITER_IMAGE, etc. The object identifier at the end of the OBJECT statement is optional, but if it appears it must match the name given at the beginning of the OBJECT statement.

12.4.4.1 Implementation of OBJECT Statements

It is recommended that all software for writing ODL should include the object identifier at the end as well as the beginning of every OBJECT statement.

12.4.5 GROUP Statement

The GROUP statement is used to group together statements which are not components of a larger object. For example, in a file containing many images, the group BEST_IMAGES might contain the object descriptions of the three highest quality images. The three image objects in the BEST_IMAGES group don't form a larger object: all they have in common is their superior quality.

The GROUP statement is also used to group related attributes of an object. For example, if two attributes of an image object are the time at which the camera shutter opened and closed, then the two attributes might be grouped as follows:

```
GROUP   = SHUTTER_TIMES
  START  = 12:30:42.177
  STOP    = 14:01:29.265
END_GROUP = SHUTTER_TIMES
```

The format of the group statement is as follows:

```
group_statement ::  =      group = group_identifier
                           [statement]*
                           end_group [= group_identifier]
```

The group identifier gives a name to the particular group, as shown in the example for shutter times above. The object identifier at the end of the GROUP statement is optional, but if it appears it must match the name given at the beginning of the GROUP statement. Groups may be nested within other groups. There is no limit to the depth to which groups can be nested.

12.4.5.1 Implementation of GROUP Statements

It is recommended that all software for writing ODL should include the group identifier at the end as well as the beginning of every GROUP statement.

12.4.5.2 PDS Usage of GROUP

Although the ODL supports the GROUP statement, the PDS does not recommend its use because of confusion concerning the difference between OBJECT and GROUP.

12.5 Values

ODL provides scalar values, ordered sequences of values, and unordered sets of values.

```
                    value :: = scalar_value | sequence_value | set_value
```

A scalar value consists of a single lexical element:

```
scalar_value :: = numeric_value |
                  date_time_value |
                  text_string_value |
                  symbol_value
```

The format and use of each of these scalar values is discussed in the sections below.

12.5.1 Numeric Values

A numeric scalar value is either a decimal or based integer number or a real number. A numeric scalar value may optionally specify a units expression.

```
        numeric_value :: =        integer [units_expression] |
                                  based_integer [units_expression] |
                                  real [units_expression]
```

12.5.2 Units Expressions

Many of the values encountered in scientific data are measurements of something. In most computer languages, only the magnitude of a measurement is represented, and not the units of measurement. The ODL, however, can represent both the magnitude and the units of a measurement. A units expression has the following format:

```
        units_expression          :: = < units_factor [mult_op units_factor] * >
        units_factor              :: = units_identifier [exp_op integer]
        mult_op                   :: = * | /
        exp_op                    :: = **
```

A units expression is always enclosed within angle brackets. The expression may consist of a single units identifier like KM (for kilometers), or SEC (for seconds). Examples are the distance 1.341E6 <KM> and the time 1.024 <SEC>. More complex units can also be represented; for example, the velocity 3.471 <KM/SEC> or the acceleration 0.414 < KM/SEC/SEC>. There is often more than one way to represent a unit of measure. For example:

- 0.414 <KM/SEC/SEC>
- 0.414 <KM/SEC**2>

- 0.414 <KM*SEC**-2>

are all valid representations of the same acceleration. The following rules apply to units expressions:

- The exponentiation operator can specify only a decimal integer exponent. The exponent value may be negative, which signifies the reciprocal of the units. For example, 60.15 < HZ> and 60.15 <SEC**-1> are both ways to specify a frequency.
- Individual units may appear in any order. For example, a force might be specified as either 1.55 <GM*CM/ SEC**2> or 1.55 <CM*GM/SEC**2>.

12.5.2.1 Implementation of Numeric Values

There is no defined maximum or minimum magnitude or precision for numeric values. In general, the actual range and precision of numbers that can be represented will be different for each kind of computer used to read or write an ODL-encoded label. Developers of software for reading/writing ODL should document the following:

- The most positive and most negative integer numbers that can be represented.
- The most positive and most negative real numbers that can be represented.
- The minimum number of significant digits which a real number can be guaranteed to have without loss of precision. This is to account for the loss of precision that can occur when representing real numbers in floating point format within a computer. For example, a 32-bit floating point number with 24-bits for the fraction can guarantee at least 6 significant digits will be exact (the seventh and subsequent digits may not be exact because of truncation and round-off errors).

If software for reading ODL encounters a numeric value that is too large to be represented, then the software shall report an error to the user.

12.5.3 Text String Values

A text string value consists of a text string lexical element:

text_string_value :: = quoted_text

12.5.3.1 Implementation of String Values

A text string read in from a label is reassembled into a string of characters. The way in which the string is broken into lines in a label doesn't affect the format of the string after it has been reassembled. The following rules are used when reading text strings:

- If a format effector or a sequence of format effectors is encountered within a text string, then the effector or sequence of effectors is replaced by a single space character, unless the last character is a hyphen (dash) character. Any spacing characters at the end of the line are removed and any spacing characters at the beginning of the following line are removed. This allows a text string in a label to appear with the left and right margins set at arbitrary points without changing the string value. For example, the following two strings are the same:

> “To be or not to be”
>
> and
>
> “To be or
> not to be"

- If the last character on a line prior to a format effector is a hyphen (dash) character, then the hyphen is removed. Any spacing characters at the beginning of the following line are removed. This follows the standard convention in English of using a hyphen to break a word across lines. For example, the following two strings are the same:

> “The planet Jupiter is very big”
>
> and
>
> “The planet Jupi-
> ter is very big”

- Control codes, other than the horizontal tabulation character and format effectors, appearing within a text string are removed.

12.5.3.1.1 PDS Text String Formatting Conventions

The PDS defines a set of format specifiers that can be used in text strings to indicate the formatting of the string on output. These specifiers can be used to indicate where explicit line breaks should be placed, and so on. The format specifiers are:

- \n - Indicates that an end-of-line sequence should be inserted.
- \t - Indicates that a horizontal tab character should be inserted.
- \f - Indicates that a page break should be inserted.
- \v - Must be used in pairs, begin and end. Interpreted as verbatim.
- \\- Used to place a backslash in a text string.

For example, the string

> “This is the first line \n and this is the second line.”

on output will print as:

```
This is the first line
and this is the second line.
```

Note that these format specifiers have meaning only when a text string is printed, and not when the string is read in or stored.

12.5.4 Symbolic Literal Values

A symbolic value may be specified as either an identifier or a symbol string:

```
symbolic-value :: = identifier | quoted_symbol
```

The following statements assign attributes to symbolic values specified by identifiers:

```
TARGET_NAME          = IO
SPACECRAFT_NAME      = VOYAGER_2
SPACECRAFT_NAME      = "VOYAGER-2"
SPACECRAFT_NAME      = "VOYAGER 2"
REFERENCE_KEY_ID     = SMITH1997
REFERENCE_KEY_ID     = "LAUREL&HARDY1997"
```

The quotes must be used if the symbolic value does not have the proper format for an identifier or if it contains characters not allowed in an identifier. For example, the value ’FILTER_+_7’ must be enclosed within quotes, since this would not be a legal ODL identifier. Similarly, the symbolic value ’U13-A4B’ must be in quotes because it contains a special character (the dash) not allowed in an identifier. There is no harm in putting a legal identifier within quotes; for example:

```
SPACECRAFT_NAME = "VOYAGER_2"
```

is equivalent to the last example above.

Symbolic values may not contain format effectors, i.e., may not cross a line boundary.

12.5.4.1 Implementation of Symbolic Literal Values

Symbolic values will be converted to upper case on input. This means that a lowercase string is converted to an equivalent uppercase string; as in the following example:

```
Original string:     SPACECRAFT_NAME          = "Voyager_2"
Converted string:    SPACECRAFT_NAME          = "VOYAGER_2"
```

12.5.4.2 PDS Recommendation on Symbolic Literal Values

Since the current use of the ODL within the PDS does not require the explicit specification of symbolic literals or symbol strings, the PDS recommends that double quotation marks (") be used instead of apostrophes.

12.5.5 Sequences

A sequence represents an ordered set of values. It can be used to represent arrays and other kinds of ordered data. Only one and two dimensional sequences are allowed.

```
sequence_value          :: = sequence_1D | sequence_2D
sequence_1D             :: = (scalar_value [, scalar_value]*)
sequence_2D             :: = ([sequence _1D] +)
```

A sequence may have any kind of scalar value for its members. It is not required that all the members of the sequence be of the same kind of scalar value. Thus a sequence may represent a heterogeneous record. Each member of a two dimensional sequence is a one dimensional sequence. This can be used, for example, to represent a table of values. The order in which members of a sequence appear must be preserved. There is no upper limit on the number of values in a sequence.

```
For example:   AVERAGE_ECCENTRICITY              = (0,1,2,3,4,5,9)
```

12.5.6 Sets

Sets are used to specify unordered values drawn from some finite set of values.

```
               set_value :: = {scalar_value [, scalar_value]*} | {}
```

Note that the empty set is allowed: The empty set is denoted by opening and closing brackets with nothing except optional spacing characters or format effectors between them.

The order in which the members appear in the set is not significant and the order need not be preserved when a set is read and manipulated. There is no upper limit on the number of values in a set.

```
For example:   FILTER_NAME            = { RED, BLUE, GREEN, HAZEL }
```

12.5.6.1 PDS Implementation of Sets

The PDS allows only symbol values and integer values within sets.

12.6 ODL Summary

Character Set (12.2)

The ODL uses the ISO 646 character set (the American version of the ISO 646 standard is ASCII). The ODL character set is partitioned as follows:

```
character            : : = letter | digit | special_character |
                           spacing_character | format_effector |
                           other_character
letter               : : = A-Z | a-z
digit                : : =  0 | 1  | 2 | 3 | 4 | 5 | 6 | 7 | 8| 9
special_character    : : =  { | } | ( | ) | + | - | . | " | ’ | = |
                              _ | , | / | * | : | # | & | ^ | < | >
spacing_character    : : = space | horizontal tabulation
format_effector      : : = carriage return | line feed |
                           form feed | vertical tabulation
other_character      :: =  ! | $ | % | ; | ? | @ | [ | ] | ` | ~ |
                           vertical bar | other control characters
```

Lexical Elements (12.3)

```
integer              : : = [sign] unsigned_integer
unsigned_integer     : := [digit]+
sign                 :: = + | -
based_integer        : : = radix # [sign] [extended_digit]+ #
extended_digit       : : = digit | letter
radix                : : = unsigned_integer
real                 : : = [sign] unscaled_real |  [sign] scaled_real
unscaled_real        : : = unsigned_integer . [unsigned_integer] |
                           . unsigned_integer
scaled_real          : : =  unscaled_real exponent
exponent             : : =E integer | e integer
date                 : : =year_doy | year_month_day
year_doy             : : =year - doy
year_month_day       : : =year - month - day
year                 : : =unsigned_integer
month                : : =unsigned_integer
day                  : : =unsigned_integer
doy                  : : =unsigned_integer
time                 : : =local_time | utc_time | zoned_time
local_time           : : =hour_min_sec
utc_time             : : =hour_min_sec Z
zoned_time           : : =hour_min_sec zone_offset
hour_min_sec         : : =hour : minute [ : second]
zone_offset          : : =sign hour [: minute]
hour                 : : =unsigned_integer
minute               : : =unsigned_integer
second               : : =unsigned_integer | unscaled_real
date_time            : : =date T time
quoted_text          : : =“[character]*"
quoted_symbol        : : =’[character]+'
identifier           : : = letter [letter | digit | _letter | _digit ]*
```

Statements (12.4)

```
label                : : = [statement]*
                                   end
statement            : : = assignment_stmt | pointer_stmt |
                        object_stmt | group_stmt
assignment_stmt   : : = attribute_identifier = value
pointer_stmt      : : = ^ object_identifier = value
object_stmt       : : = object = object_identifier
                             [statement]*
                         end_object [= object_identifier]
group_stmt        : : = group = group_identifier
                             [statement]*
                        end_group [= group_identifier]
```

Values (12.5)

```
value             : : = scalar_value | sequence_value |     set_value
scalar_value      : : = numeric_value | date_time_value |
                         text_string_value | symbolic_value
numeric_value     : : = integer [units_expression] |
                         based_integer [units_expression] |
                         real [units_expression]
units_expression  : : =<units_factor[mult_op units_factor]* >
units_factor      : : = units_identifier [exp_op integer]
mult_op           : : = * | /
exp_op            : : = **
date_time_value   : : = date | time  | date_time
text_string_value : : = quoted_text
symbolic_value    : : =  identifier | quoted_symbol
sequence_value    : : =sequence_lD | sequence_2D
sequence_1D       : : = (scalar_value [, scalar_value]*)
sequence_2D       : : = ([sequence_lD]+)
set_value         : : = { scalar_value [,scalar_value]* } | { }
```

12.7 Differences Between ODL Versions

This appendix summarizes the differences between the current Version 2 of ODL and the previous Versions 0 and 1. Software can be constructed to read all three versions of ODL. However, it is important that software for writing labels only write labels that conform to ODL Version 2.

12.7.1 Differences from ODL Version 1

Version 1 labels were used on the Voyager to the Outer Planets CD-ROM disks and many other data sets. Version 1 did not include the GROUP statement and it had a more restrictive definition for sets (which were limited to integer or symbolic literal values) and for sequences (which were limited to arrays of homogeneous values). The following sections detail non-compatible differences and how they can be handled by software writers.

12.7.1.1 Ranges

Version 1 of the ODL had a specific notation for integer ranges:

```
range_value :: = integer..integer
```

This notation is not allowed in ODL Version 2. A parser may still recognize the ’double-dot' range notation. On output, a range shall be encoded as a two value sequence, with the low-value of the range being the first element of the sequence and the high-value being the second element of the sequence.

12.7.1.1.1 Delimiters In Sequences and Sets

The individual values in sets and sequences could be separated by a comma or by a spacing character. In Version 2, a comma is required. A parser can allow spacing characters between values as well as commas. Software that writes ODL should place commas between all values in a sequence or set.

12.7.1.1.2 Exponentiation Operator in Units Expressions

In Version 1 of the ODL the circumflex character (^) was used as the exponentiation operator in units expressions rather than the two-asterisk sequence (**). Parsers may still allow the circumflex to appear within units expressions as an exponentiation operator. Software for writing ODL should use only the ** notation.

12.7.2 Differences from ODL Version 0

Version 0 of ODL was developed for and used on the PDS Space Science Sampler CD-ROM disks. The major aspect of Version 0 is that it did not provide the OBJECT statement: all of the attributes specified in a label described a single object - namely the file that contained the label (or that was referenced by a pointer).

12.7.2.1 Date—Time Format

ODL Version 0 was produced prior to the space community's acceptance of the ISO/DIS 8601 standard for dates and time and it uses a different date and date-time format. The format for Version 0 dates and date-times is as follows:

```
date          :: = year / month / day_of_month  | year / day_of_year
date_time     :: = date - time zone
zone          :: = < identifier>
```

The definition of time in ODL Version 0 was a subset of ODL Version 2; therefore parsers that handle Version 2 time formats will also handle Version 0 times. Software for writing ODL must output dates and date-times in the Version 2 format only.

12.7.3 ODL/PVL Usage

A concept for a Parameter Value Language/Format (PVL) is being formalized by the Consultative Committee for Space Data Systems (CCSDS). It is intended to provide a human readable data element/value structure to encode data for interchange. The CCSDS version of the

PVL specification is in preliminary form.

Some organizations which deal with the PDS have accepted PVL as their standard language for product labels. Largely because PVL is a superset of ODL, some PVL constructs are not supported by the PDS. In addition, some ODL constructs may be interpreted differently by PVL software.

The ODL/PVL usage standard defines restrictions on the use of ODL/PVL in archive quality data sets. These restrictions are intended to ensure the compatibility of PVL with the Object Description Language (ODL) and existing software.

1. Labels constructed using PVL may be attached, embedded in the same file as the data object it describes, or detached, residing in a separate file and pointing to the data file the label describes.

2. All statements shall be terminated with a <CR> <LF> pair. Semicolons shall not be used to terminate statements.

3. Only alphanumeric characters and the underscore character shall be used in data elements and undelimited text values (literals). In addition, data element and undelimited text values must begin with a letter.

4. Keywords shall be 30 characters or less in length.

5. Keywords and standard values shall be in upper case. Literals and strings may be in upper case, lower case, or mixed case.

6. Comments shall be contained on a single line, and a comment terminator (*/) shall be used. Comments shall not be embedded within statements. Comments shall not be used on the same line as any statement if the comment precedes the statement. Comments may be on the same line as a statement if the comment follows the statement and is separated from the statement by at least one white space, but this is not recommended.

7. Text values that cross line boundaries shall be enclosed in double quotation marks (“ ”).

8. Values that consist only of letters, numbers, and underscores (and that begin with a letter) may be used without quotation marks. All other text values must be enclosed in either single (’ ’) or double (“ ”) quotation marks.

9. Sequences (arrays) shall be limited to 2 dimensions. NULL (empty) sequences are not allowed. Sets shall be limited to one dimension. In other words, sets and sequences shall not be used inside a set.

10. Only the OBJECT, END_OBJECT, GROUP and END_GROUP aggregation mark-

ers shall be used.

11. Units expression shall only be allowed following numeric values (e.g, “DATA_ELEMENT = 7 <BYTES>” is valid. but “DATA_ELEMENT = MANY <METERS>” is not.

12. Units expression shall include only alphanumeric characters, the underscore, and the symbols *,/,(,), and **. (The last represents exponentiation).

13. Signs shall not be used in non-decimal numbers. (e.g., “2#10001#” is valid, but “-2#10001#” and “2#-10001#” are not.) Only the bases 2,8, and 16 shall be used in non-decimal numbers.

14. Alternate time zones (e.g., YYYY-MM-DDTHH:MM:SS.SSS + HH:MM) shall not be used. Only the format YYYY-MM-DDTHH:MM:SS.SSS shall be used.

15. Always provide all digit positions in dates and times. Zeros shall be used to replace missing digits.

16. An END statement shall be included at the end of the ODL/PVL statement list.

The following are guidelines for formatting ODL/PVL expressions.

1. The assignment symbol (=) shall be surrounded by blanks.

2. Assignment symbols (=) should be aligned if possible.

3. Keywords placed inside an aggregator (OBJECT or GROUP) shall be indented with respect to the OBJECT and END_OBJECT or GROUP and END_GROUP statements which enclose them.

4. PDS label lines shall be 80 characters or less in length, including the end-of-statement <CR> <LF> delimiter. While 80 characters can be displayed on most screens, some editors and databases will wrap or truncate lines that exceed 72 characters.

5. TABs shall not be used in PDS Labels. Although both ODL and PVL allow the use of TABs some simple parsers cannot handle them. Use spaces instead.

Chapter 13

PDS Objects

The Planetary Data System has designed a set of standard objects to be used for submitting catalog object templates as well as for labeling data products. These standard objects, along with definitions of individual keywords comprising those objects, are defined in the *Planetary Science Data Dictionary*. In addition, object definitions and examples are also included as Appendix A and Appendix B of this document.

13.1 Generic and Specific Data Object Definitions

For each type of data object that PDS has defined (i.e., IMAGE, TABLE, etc.), there are two categories, generic and specific. A generic object is the universal definition of an object, or superset of keywords that can be used. A specific object is a subset used for a specific data product to allow effective use of validation tools.

Generic objects are designed and approved by the Planetary Data System. The elements used to define objects are classified either as Required or Optional. The Required and Optional member elements are explicitly listed while the Optional member elements may include any element in the data dictionary. A Specific object is defined for a particular data product and is based in a selected Generic object. All Required elements and selected Optional elements from the Generic object are used to define the Specific object.

Using the generic object definition as a guide and consulting with a Central Node Data Engineer, a user may then customize the object by first using all the required keywords, and then choosing which optional keywords apply to the data product. In addition, any keywords listed in the *Planetary Science Data Dictionary* can be chosen for special purposes. The resulting object will be a specific object that is subject to approval during a design review.

The following examples illustrate the migration from the generic IMAGE object to a specific IMAGE object and then an instance of that specific IMAGE. Note that when a specific case is used, that usage should be consistent for all labels defining a like data product.

```
OBJECT                                        = GENERIC_OBJECT_DEFINITION
NAME                                          = IMAGE
STATUS_TYPE                                   = APPROVED
STATUS_NOTE                                   = "V2.1  1991-01-20  MDM  New Data Object Definition"
DESCRIPTION                                   =  "An image object is a regular array of sample values.  Image
objects are normally processed with special display tools to produce a visual representation of the sample   values.  This is done
by assigning brightness levels or display   colors to the various sample values. Images are composed of LINES   and SAMPLES.
They may contain multiple bands, in one of several  storage orders.

Note:  Additional engineering values may be prepended or appended to each LINE of an image, and are stored as concatenated
TABLE objects, which must be named LINE_PREFIX and LINE_SUFFIX.  IMAGE objects may be associated with other
```

```
objects, including HISTOGRAMs, PALETTEs, HISTORY and TABLEs which contain statistics, display parameters,
engineering values or other ancillary data."

SOURCE_NAME                          = "PDS CN/M.Martin"
REQUIRED_ELEMENT_SET                 = {LINE_SAMPLES, LINES, SAMPLE_BITS,
                                        SAMPLE_TYPE}
OPTIONAL_ELEMENT_SET                 = {BAND_SEQUENCE, BAND_STORAGE_TYPE,
                                        BANDS, CHECKSUM, DERIVED_MAXIMUM,
                                        DERIVED_MINIMUM, DESCRIPTION,
                                        ENCODING_TYPE, FIRST_LINE,
                                        FIRST_LINE_SAMPLE, INVALID,
                                        LINE_PREFIX_BYTES, LINE_SUFFIX_BYTES, MISSING,
                                        OFFSET, SAMPLE_BIT_MASK, SAMPLING_FACTOR,
                                        SCALING_FACTOR, SOURCE_FILE_NAME,
                                        SOURCE_LINES,  SOURCE_LINE_SAMPLES,
                                        SOURCE_SAMPLE_BITS, STRETCHED_FLAG,
                                        STRETCH_MAXIMUM, STRETCH_MINIMUM, PSDD}
REQUIRED_OBJECT_SET                  = “N/A”
OPTIONAL_OBJECT_SET                  = “N/A”

OBJECT_CLASSIFICATION_TYPE           = STRUCTURE

OBJECT                               = ALIAS
NAME                                 =  “N/A”
USAGE_NOTE                           = “N/A”
END_OBJECT                           =  ALIAS

END_OBJECT                           = GENERIC_OBJECT_DEFINITION
```

This next example illustrates IMAGE object definition being used for a specific case.

```
OBJECT                          = SPECIFIC_OBJECT_DEFINITION
NAME                            = XYZ_IMAGE
STATUS_TYPE                     = APPROVED
STATUS_NOTE                     = "V2.1 1991-02-10  TMA New specific data object definition"
DESCRIPTION                     = "The XYZ image is..."

SOURCE_NAME                     = "PDS CN/M.Martin"
REQUIRED_ELEMENT_SET            = {LINE_SAMPLES, LINES, SAMPLE_BITS,
                                     SAMPLE_TYPE, SAMPLING_FACTOR,
                                     SOURCE_FILE_NAME,
                                     SOURCE_LINES,  SOURCE_LINE_SAMPLES,
                                     SOURCE_SAMPLE_BITS, FIRST_LINE,
                                     FIRST_LINE_SAMPLE}

OBJECT_CLASSIFICATION_TYPE      = STRUCTURE

OBJECT                          = ALIAS
NAME                            = “N/A”
USAGE_NOTE                      = “N/A”
END_OBJECT                      = ALIAS

END_OBJECT                      = SPECIFIC_ OBJECT_DEFINITION
```

13.2 Primitive Objects

Generic objects have a subclass called primitive objects that include ARRAY, COLLECTION, ELEMENT, and BIT_ELEMENT. A primitive object is primarily used as the foundation for defining the elementary structure of PDS objects that have either more abstract or more uncommon layouts than more common structures like TABLES or IMAGEs. For example, a simple camera image abstractly described by a PDS IMAGE object, shown in Example 1, could alternately be described using a 2-dimensional ARRAY object, as shown in Example 2.

Example 1

```
OBJECT          = IMAGE
LINES           = 800
LINE_SAMPLES    = 600
. . .
END_OBJECT      = IMAGE
```

Example 2

```
OBJECT          = ARRAY
AXES            = 2
AXIS_ITEMS      = (800, 600)
AXIS_NAME       = (LINES, LINE_SAMPLES)
                  . . .
END_OBJECT      = ARRAY
```

However, given the PDS objective of defining a robust object model for planetary science data, it is recommended that primitive objects only be used when other PDS objects result in a misleading or incorrect description of the data being labeled.

Chapter 14

Pointer Usage

Within PDS labels, pointers are used to indicate the locations of objects within the same file or references to external files. A pointer statement is indicated in a PDS label or catalog object by an ASCII caret (^).

14.1 Types of Pointers

Pointer statements fall into three main categories: data location pointers, include pointers, and related information pointers.

14.1.1 Data Location Pointers (Data Object Pointers)

The most common use of pointers occurs in PDS labels to link together data object descriptions with the actual data. The syntax for the values of these pointers depends on whether the label is attached or detached from the data it describes. Examples of these data location pointer statements are:

```
(1)     ^IMAGE          = 12
(2)     ^IMAGE          = 600 <BYTES>

(3)     ^INDEX_TABLE    = "INDEX.TAB"
(4)     ^SERIES         = ("C100306.DAT", 2)
(5)     ^SERIES         = ("C100306.DAT", 700 <BYTES>)
```

The first and second examples illustrate pointers in attached labels. This type of pointer allows reading software to scan the label for the appropriate pointer, and then skip right to the data at its location elsewhere in the file. In the first example, the data begin at record 12 of the labeled file. In the second example, the data begin at byte 600 of the labeled file.

In examples 3 through 5, external data files are referenced. As these pointers occur in detached labels, they must identify a file name, and if the data do not begin at record 1 of the data file, a location as well. In example 3, the data begin at record 1 of the data file "INDEX.TAB". In example 4, the data begin at record 2 of the data file, "C100306.DAT". In example 5, the data begin at byte 700 of the data file.

14.1.2 Include Pointers

Another common use of pointers occurs in PDS labels or completed catalog templates that reference external files to be included directly at the location of the pointer statement. These are classified as 'include' type pointers since they act like #INCLUDE statements in C program source files. Pointers with the class names of STRUCTURE, CATALOG, and MAP_PROJECTION fall into this category. As illustrated below, include files contain only PDS data object definitions or completed catalog object templates.

Examples of include pointer statements are:

```
(1)       ^STRUCTURE  = "ENGTAB.FMT"
(2)       ^STRUCTURE  = "IMAGE.FMT"
(3)       ^CATALOG  = "CATALOG.CAT"
(4)       ^DATA_SET_MAP_PROJECTION  = "DSMAPDIM.CAT"
```

In the first example, an external structure file is referenced from a TABLE object. The file ENGTAB.FMT contains the column object definitions needed to complete the TABLE object. In cases such as this, column objects would be stored in a separate file if the table is especially large (with many columns), making its label unwieldy, or if the file containing column objects can be referenced by more than one label through the use of the pointer.

In the second example, the structure of an image (i.e., all statements beginning with the OBJECT = IMAGE statement and ending with the END_OBJECT = IMAGE statement) is defined in an external file called IMAGE.FMT.

In the third example, the external file, CATALOG.CAT, is pointed to from the VOLUME object in order to provide a full set of catalog information associated with the volume.

In the fourth example, the external file, DSMAPDIM.CAT, is referenced in the IMAGE_MAP_PROJECTION object to complete the map projection information associated with the image.

14.1.3 Related Information Pointers (Description Pointers)

The last type of use of pointer statements occurs in PDS labels that reference external files that provide additional documentation that may be of special use to a human reader of the label. These files are indicated by the DESCRIPTION or DESC class words, and reference text files that are not written in ODL. This pointer is not meant to refer to software tools.

An example of a description pointer statement is:

```
          ^DESCRIPTION   = "TRK_2_25.ASC"
```

In this example, the pointer references a PDS-labeled external ASCII document file, TRK_2_25.ASC, which provides a detailed description of the data.

14.2 Rules for Resolving Pointers

The following set of rules exist for resolving pointer statements that reference external files:

For any pointer statement in FILE_A,

(1) Look in the same directory as FILE_A

(2a) For a single physical volume (no logical volumes), look in the following top level directory:

Pointer	Directory
^STRUCTURE	LABEL
^CATALOG	CATALOG
^DATA_SET_MAP_PROJECTION	CATALOG*
^INDEX_TABLE	INDEX
^DESCRIPTION or ^TEXT	DOCUMENT

(2b) Within a logical volume, look in the top level subdirectory specified by the LOGICAL_ VOLUME_PATH_NAME keyword:

Pointer	LOGICAL_VOLUME_PATH_NAME / Directory
^STRUCTURE	LABEL
^CATALOG	CATALOG
^DATA_SET_MAP_PROJECTION	CATALOG*
^INDEX_TABLE	INDEX
^DESCRIPTION or ^TEXT	DOCUMENT

* Note: For volumes using PDS Version 1 or 2 standards, the MAP_PROJECTION files may be located in the LABEL directory

All pointers to data objects should be resolved in step (1), since these files are always required to be located in the same directory as the label file.

Chapter 15

Record Formats

The choice of the proper record format is determined by the applications which the data will support. In general, fixed length records are well-suited to the storage of binary data files, such as images, binary tables or qubes. These files are expected to be transported and used in structured environments. They shall also be used for ASCII tables to promote transportability. Input/output operations with FIXED_LENGTH files will use read and write statements which read RECORD_BYTES number of bytes with each operation.

Variable length files are less transportable and require special software to read. Their use is discouraged except in instances where they may optimize storage efficiency or access. An example of such an application is the compressed image format being used for CD-ROM storage.

For CD-ROMs that are meant to be VAX/VMS-compatible (ie., for CDs with XARs), it is recommended that all records in fixed length or variable length files contain an *even* number of bytes. Thus records which contain an odd number of bytes would be padded by one byte to give them an even length.

Stream records should be used for text files for ease of transportation to different computer systems. Input/output operations with stream files will generally use string-oriented access, retrieving a record from the file each time.

Table 15.1: Recommended Record Formats

	RECORD_TYPE=FIXED	RECORD_TYPE=STREAM	RECORD_TYPE=VARIABLE
Data format	BINARY, ASCII	ASCII	BINARY
Environment	STRUCTURED	ADHOC	VERY STRUCTURED
Data volume	LARGE	SMALL, MEDIUM	VERY LARGE
Input / Output	READ / WRITE	STRING I/O	CUSTOM, SPICE

15.1 Fixed Length Record Formats

Fixed length record formats normally use a physical record length (RECORD_BYTES) which corresponds directly to the logical length of the data objects (that is, one physical record for each image line, or one physical record for each row of a table). In some cases, logical records are *blocked* into larger physical records to provide more efficient storage and access to the data. This blocking is still an important consideration when storing data on magnetic tape, (which requires a gap on the tape between records), but is not generally a consideration in data sets stored on magnetic or CD-ROM disks. In other cases, the physical record length is arbitrary, and only specifies a unit of data for input/output operations, as in FITS format files or USGS PICS images.

The use of a record length which matches the size of the primary data object in a file is recommended, to provide fairly simple file access with a variety of applications. In this approach, objects within a file are all stored in physical records of RECORD_BYTES length. Figure 15.1 illustrates the physical and logical structure used to build a standard PDS FIXED_LENGTH file.

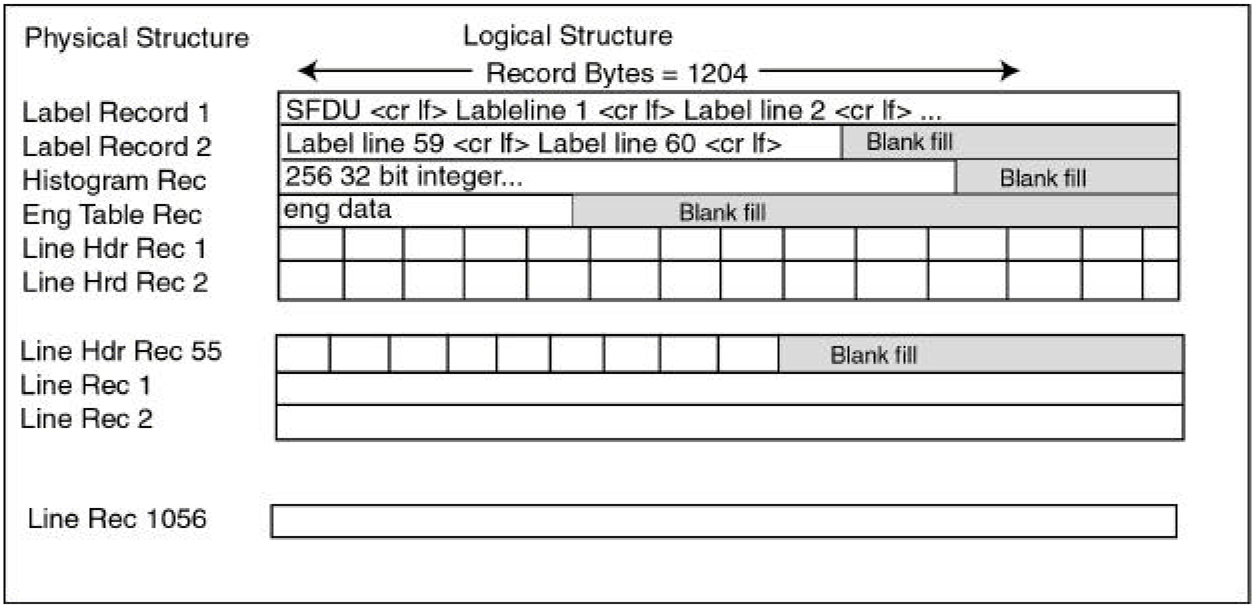


Figure 15.1 Physical and Logical Structure for Fixed Length Files

15.2 Stream Record Formats

Stream records consist of ASCII text delimited with a carriage return (CR) and line feed (LF) sequence. Different computers interpret these codes differently. For example, IBM PC's use the two-byte CR/LF sequence to terminate a line of text. UNIX systems use only a line feed. The Macintosh uses only a carriage return. VAX computers support these various formats as stream files, but prefer to store text files internally as variable length records.

Despite the confusion, stream files can easily be transmitted via text-oriented communications facilities like NASAMAIL, Eudora, or QuickMail. In addition, most file transfer protocols (KERMIT, FTP) will automatically make the needed conversions when stream files are transported between different computers.

PDS has adopted the CR/LF as the standard line delimiter for archival products. Note, in particular, that CR/LF is the required line terminator for all PDS labels and catalog files. This is the only end-of-line sequence that insures that text file will be viewable on all computer systems.

System utilities are available on the various computer types to convert this format to the internal format if necessary.

Macintosh - Apple File Exchange, MS-DOS to Mac option.
Unix - Translate utility (tr-d'\15' <input_file>output_file)

The stream format is recommended for the transfer and archive of text and for files containing detached labels. While stream format can be used for ASCII tables, it is recommended that the FIXED_LENGTH format be used when storing these tables on archival or distributable media (CD-ROM).

15.3 Variable Record Formats

A third category of record type is variable length. The use of variable length records is discouraged, since they are operating-system dependent. They should only be used in the following circumstances:

- Software that can operate on a variety of hosts is provided along with the data. For example, the Voyager CD-ROM disks contain variable length compressed images, along with a decompression program for VAX, PC, Macintosh and UNIX systems. These programs will reformat the data to a variety of user-selectable formats.
- The files are only intended for use on one computer system. For example, the Viking IRTM CD-ROM utilizes VAX/VMS variable length formats for software and command files because the software cannot be used unless it is in this format.

PDS data files using variable length records shall follow the VAX/VMS conventions where the records are preceded by a 2-byte (LSB first or swapped) integer which defines the length of the record with no carriage control. The reason for this choice is that VAX/VMS supports variable length records and numerous planetary science data files are stored in this format.

15.4 Undefined Record Formats

Undefined record formats are those which have no implied record structure. For files with attached labels, the label portion should be written using undefined record format and should use record terminators as in the stream case. When data are written using undefined format, no record terminators or specific record length is implied; it is assumed to be a stream of bytes. It is recommended that fixed length records rather than undefined record format be used whenever possible.

15.5 Detached Label Files

Detached label files should be in stream record format. The data elements in a detached label ALWAYS REFER TO THE DATA FILE, not to the detached label file. Thus a RECORD_TYPE = FIXED_LENGTH data element in a label file refers to the record type of the data file, not the label file itself. Detached label files shall carry the file extension “.LBL” so that they can be easily identified by users.

Chapter 16

SFDU Usage

The SFDU Usage Standard defines restrictions on the use of Standard Formatted Data Units (SFDUs) in archive quality data sets. **PDS does not require that data products are packaged as SFDUs. However, if data products are packaged as SFDUs, the following standards are in effect.**

A recommendation for the standardization of the structure and construction rules of SFDUs for the interchange of digital space-related data has been prepared by the Consultative Committee for Space Data Systems (CCSDS). An SFDU is a *type-length-value* object. More simply stated, each SFDU consists of a type identifier which indicates the type of data within the SFDU, a length field which either states the length of the data or indicates how the data are delimited, and a value field which is the data itself. Both the type and the length fields are included in a 20 byte label which will be called an *SFDU label* in this document. The value field immediately follows the 20 byte SFDU Label. For PDS data products, the value field contains the PDS label including one or more data object definitions (such as an image).

There are three versions of SFDUs. In Version 1, the length of an SFDU was represented in binary. In Version 2, the length could also be represented in ASCII. In Version 3, the length can be represented in binary, ASCII, or using one of several delineation techniques. Unless previously negotiated, all PDS data products packaged as SFDUs shall be constructed using Version 3 SFDU Labels.

A Version 3 SFDU label consists of the following parts:

l)	Control Authority ID	4 Bytes
2)	Version ID	1 Byte
3)	Class ID	1 Byte
4)	Delimiter Type	1 Byte
5)	Spare	1 Byte
6)	Description Data Unit ID	4 Bytes
7)	Length	8 Bytes

The Control Authority ID and the Description Data Unit ID together form an identifier called an Authority and Description Identifier which points to a semantic (Planetary Science Data Dictionary) and syntactic (Object Definition Language, 2.0) description of the value field.

Version 3 allows delimitation of SFDUs by end-of-file or by start markers and end markers rather than by explicit byte counts. Further details of the SFDU architecture will not be discussed here. Other sources of information can be found in the *SFDU References* listed in the *Introduction* to this document.

Since archive quality data sets are internally defined, only a limited set of SFDU labels are used to identify the files on a data volume. The full suite of available SFDU classes is not used in the packaging of PDS data products. The PDS has adopted this philosophy in order to simplify not only the archive products themselves, but also the software processing of those products. PDS labels are included in the data products, and the information in these PDS Labels is considered more than adequate for data identification and scientific analysis.

The standard usage of SFDUs by PDS in current missions and data restoration is different than the usage of SFDUs in data products from upcoming missions fully supported by the JPL Advanced Multi-Mission Operations System (AMMOS). The following sections define the standard usage of SFDUs for each source of data.

Two SFDU organizations are allowed in PDS data products. The first organization (the ZI Structure) has been used historically in PDS data products from restoration and past missions. The second organization (the ZKI organization) is required for data products which pass through the JPL Advanced Multi-Mission Operations System (AMMOS) Project Database.

16.1 The ZI SFDU Organization

Any PDS data products that are packaged as SFDUs and are not required to pass through the AMMOS Project Database as part of an active mission may use the following SFDU organization.

Each instance of a data product (file) in a data set shall include two (and only two) SFDU labels. These are a Z Class SFDU label and an I Class SFDU label. The two SFDU labels are concatenated (i.e. Z, then I) and left justified in the first line or record of the PDS label for each data product. (See Figure 16.1.) In the case of data products with detached PDS labels, the two SFDU labels shall appear in the first record of the PDS label files and no SFDU labels appear in the data object files. (See Figure 16.2.)

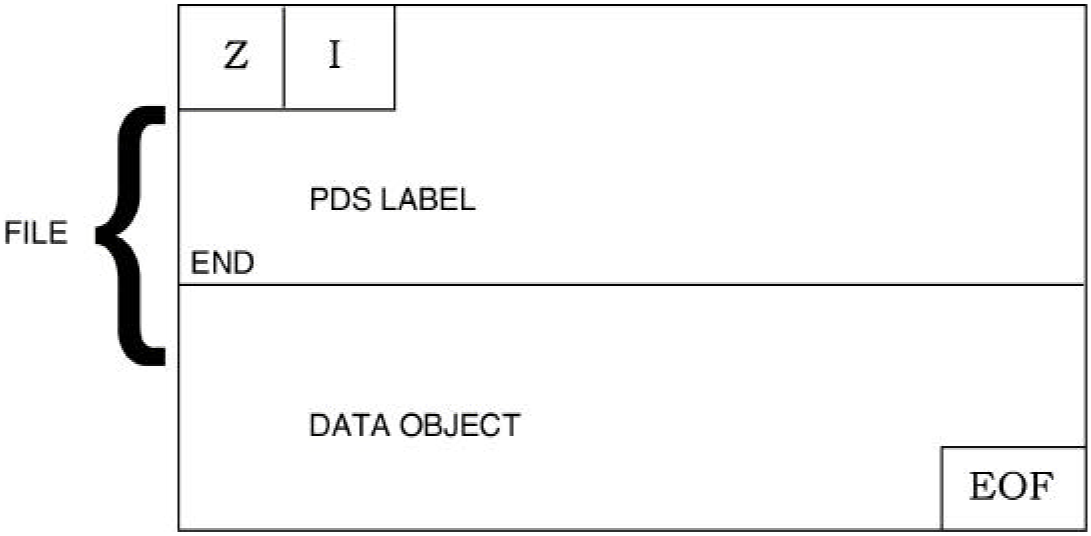


Figure 16.1 Attached PDS Label Example for non-AMMOS compatible products

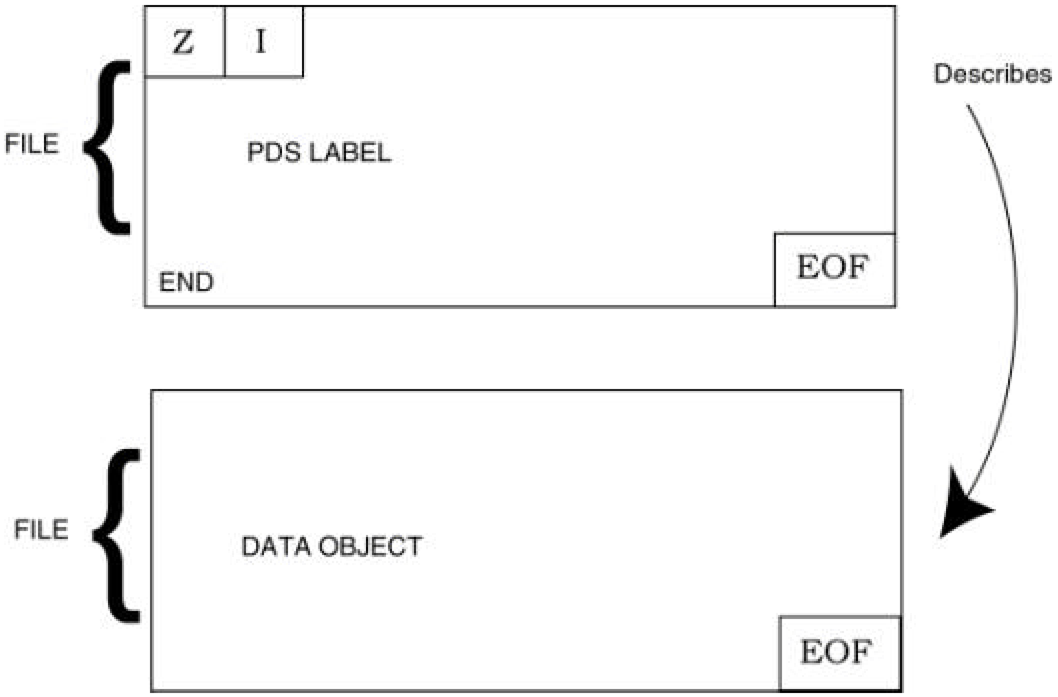


Figure 16.2 Detached PDS Label Example for non-AMMOS compatible products

The first SFDU label shall be a Z Class Version 3 SFDU label. The Z Class indicates that the value field (everything after the first 20 bytes) is an aggregation. In this case, the aggregation consists of only the I Class SFDU. This label also indicates that the delimiter type is End-of-File and that this SFDU (data product) is terminated by a single End-of-File. It shall be formed as follows:

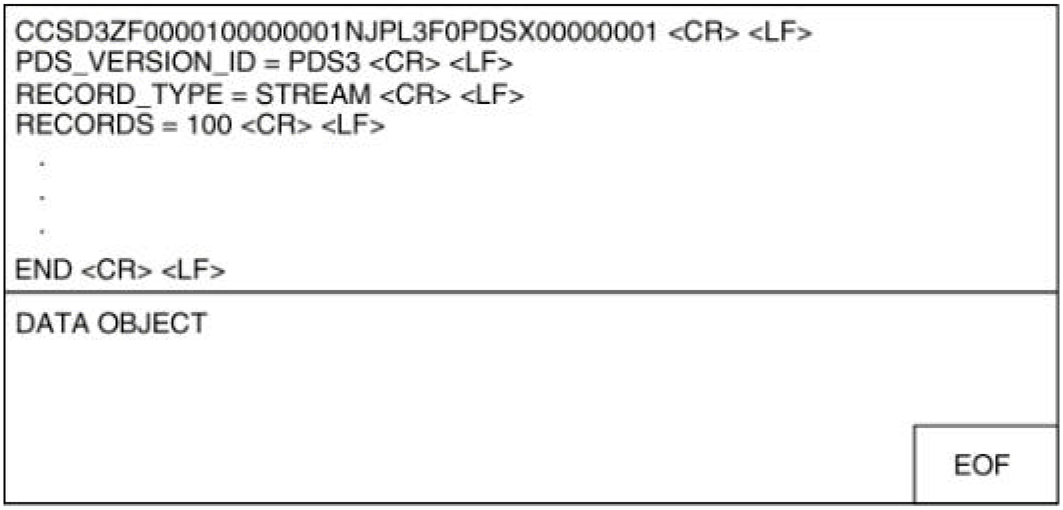
1)	Control Authority ID	CCSD
2)	Version ID	3
3)	Class ID	Z
4)	Delimiter Type	F
5)	Spare	0
6)	Description Data Unit ID	0001
7)	Length Field	00000001

Example: CCSD3ZF0000l0000000l

The second SFDU label shall be an I Class Version 3 SFDU label. Class I indicates that the value field (everything after the second 20 bytes) is application data, the PDS label and the data object(s). The Data Description Unit ID of PDSX indicates that the data product uses the Object Description Language (ODL) syntax and the Planetary Science Data Dictionary semantics to present data descriptive information. This SFDU label also indicates that the SFDU (data products) will be terminated by a single End-of-File. It shall be formed as follows:

1)	Control Authority ID	NJPL
2)	Version ID	3
3)	Class ID	I
4)	Delimiter Type	F
5)	Spare	0
6)	Description Data Unit ID	PDSX
7)	Length Field	00000001

Example: NJPL3IF0PDSX0000000l



```
CCSD3ZF0000100000001NJPL3F0PDSX00000001 <CR> <LF>
PDS_VERSION_ID = PDS3 <CR> <LF>
RECORD_TYPE = STREAM <CR> <LF>
RECORDS = 100 <CR> <LF>
  .
  .
  .
END <CR> <LF>
DATA OBJECT
                                                   EOF
```

Figure 16.3: SFDU Example

The two SFDU labels shall be concatenated, left justified, in the first line or record of the PDS label. Note that there are no characters between the two SFDU labels. See Figure 16.3.

For RECORD_TYPE = STREAM or FIXED_LENGTH or UNDEFINED, the concatenated SFDU labels shall be followed immediately by <CR><LF>. For data products that have RECORD_TYPE =VARIABLE_LENGTH, the two SFDU labels shall not be followed by <CR><LF>.

STREAM example	CCSD3ZF0000l0000000lNJPL3IF0PDSX0000000l <CR><LF>
FIXED_LENGTH Example	CCSD3ZF0000l0000000lNJPL3IF0PDSX0000000l<CR><LF>
VARIABLE_LENGTH Example	CCSD3ZF0000l0000000lNJPL3IF0PDSX0000000l
UNDEFINED Example	CCSD3ZF0000l0000000lNJPL3IF0PDSX0000000l<CR><LF>

The remainder of the PDS label begins on the next line or record. The last line of the PDS label contains the END statement. Then, if the PDS Label is attached, the data object begins on the next record. If the PDS label is detached, the END statement is the last line of the file.

16.2 The ZKI SFDU Organization

Any PDS data products that are packaged as SFDUs and are required to pass through the AMMOS Project Database as part of an active mission must use the following SFDU organization. All data products of this type are assumed to have attached PDS labels.

Each instance of a data product (file) in a data set shall include four (and only four) SFDU labels. These are the Z Class SFDU label, the K Class SFDU label, the End-Marker label for the K Class SFDU, and the I Class SFDU label. The Z and K Class SFDU labels are concatenated (i.e. Z, then K) and left justified in the first line or record of the PDS label for each data product. The End-Marker for the K Class SFDU label and the I Class SFDU label are right justified on the last record of the PDS label (following the END statement). See Figure 16.4.

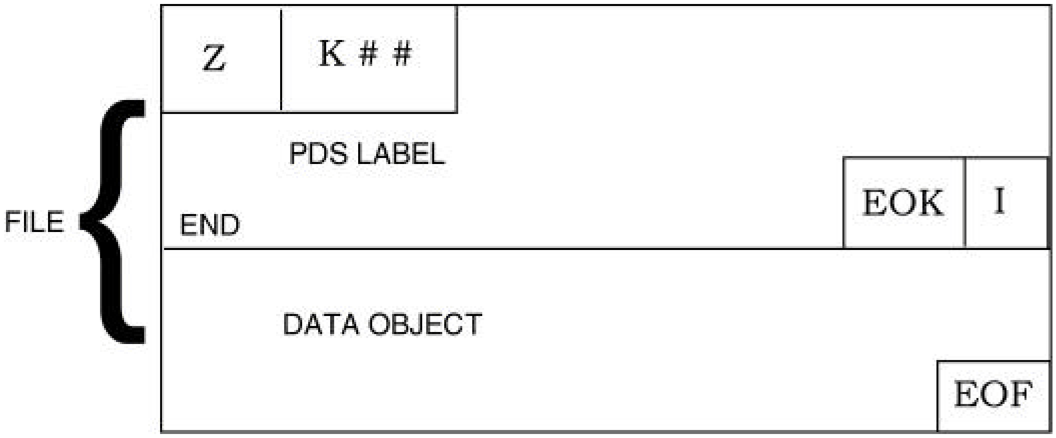


Figure 16.4: PDS Label Example for AMMOS compatible products

The first SFDU label shall be a Z Class Version 3 SFDU label. The Z Class indicates that the value field (everything after the first 20 bytes) is an aggregation. In this case, the aggregation consists of a K Class (PDS label) and an I Class (data object) SFDU. This label also indicates that the delimitation type is End-of-File and that this SFDU (data product) is terminated by a single End-of-File. It shall be formed as follows:

1)	Control Authority	CCSD
2)	Version ID	3
3)	Class ID	Z
4)	Delimiter Type	F
5)	Spare	0
6)	Description Data Unit ID	0001
7)	Length Field	00000001

Example: CCSD3ZF0000l0000000l

The second SFDU label shall be an K Class Version 3 SFDU label. Class K indicates that the value field (everything after the second 20 bytes) is catalog and directory information, i.e., the PDS label (sometimes referred to as the K Header). The Data Description Unit ID of PDSX indicates that the PDS label uses the Object Description Language (ODL) syntax and the Planetary Science Data Dictionary semantics to present data descriptive information. The SFDU label also indicates that the SFDU is delimited by a Start-Marker/End-Marker pair. It shall be formed as follows:

1)	Control Authority ID	NJPL
2)	Version ID	3
3)	Class ID	K
4)	Delimiter Type	S
5)	Spare	0
6)	Description Data Unit ID	PDSX
7)	Length Field	##mark##

The marker pattern (##mark## in the example) can be set to any String which is unlikely to be repeated elsewhere in the data product.

EXAMPLE: NJPL3KS0PDSX##mark##

The two SFDU labels shall be concatenated, left justified, in the first line or record of the PDS label. Note that there are no characters between the two SFDU labels. For data products that have RECORD_TYPE equal to VARIABLE_LENGTH the two concatenated SFDU labels shall not be followed by <CR><LF>.

EXAMPLE: CCSD3ZF0000l0000000lNJPL3KS0PDSX##mark##

The remainder of the PDS label begins on the next line. The last line of the PDS label contains the END statement. Then, in the same line or record, right justified, is the End-Marker for the K Class SFDU and the I Class SFDU label. The End-Marker pattern shall appear as:

EXAMPLE: CCSD$$MARKER##mark##

Note that the start marker and the end marker fields must be identical within the SFDU (in the example, ##mark##). Next shall be an I Class Version 3 SFDU label. Class I indicates that the value field (everything after the SFDU label) is application data, the data object. The Data Description Unit ID varies by data product type, is supplied by the JPL Control Authority, and is usually documented in the science data product Software Interface Specifications (SIS). The SFDU label also indicates that the SFDU will be terminated by a single End-of-File. It shall be formed as follows:

1)	Control Authority ID	NJPL
2)	Version ID	3
3)	Class ID	I
4)	Delimiter Type	F
5)	Spare	0
6)	Description Data Unit ID	XXXX
7)	Length Field	00000001

EXAMPLE: NJPL3IF001060000000l

where XXXX has been replaced by 0106.

The two SFDU labels shall be concatenated, right justified, and appear in the last line or record of the PDS label following the END statement. (If it happens that there is not 40 bytes left in the last record of the PDS label, add an additional record and right justify the two SFDU labels.) Note that there are no characters between the two SFDU labels, and that the marker pattern and I Class SFDU Labels are transparent to the PDS label processing software (the PDS Toolbox).

Example: END CCSD$$MARKER##mark##NJPL3IF001060000000l

The data object begins on the next physical record.

- Example for STREAM record type

End Statement blank(s) End marker I Class SFDU End of record

END CCSD$$MARKER##mark##NJPL3IF0010600000001<CR><LF>

- Example for FIXED_LENGTH record type:

End Statement Terminator Record Boundary

END <CR><LF> bbbbb CCSD$$MARKER##mark##NJPL3IF0010600000001

- Example for UNDEFINED record type:

Statement terminator

End Statement
END<CR><LF> CCSD$$MARKER##mark##NJPL3IF0010600000001

- Example for VARIABLE_LENGTH RECORD_TYPE:

Record Length   END end of statement

END   CCSD$$MARKER##mark##NJPL3IF0010600000001

16.3 Exceptions to this Standard

Software files and document files should not be packaged as SFDUs. Previous versions of the PDS standards expressed the ZI SFDU labels as an ODL statement. The ZI SFDU labels were followed by "= SFDU_LABEL".

EXAMPLE: CCSD3ZF0000100000001NJPL3IF0PDSX00000001 = SFDU_LABEL

Chapter 17

Usage of N/A, UNK and NULL

17.1 Interpretation of N/A, UNK, and NULL

During the completion of data product labels or catalog templates, it often occurs that a value is not available for a required data element. The symbolic literals "N/A", "UNK", and "NULL" are used in such cases to represent the fact that no value is available and also to suggest the reason why the value is not available. This chapter provides both descriptive and technical definitions for these symbolic literals.

The symbolic literals "N/A", "UNK", and "NULL" are allowed for use in all domains of all data elements. In the descriptions, the actual use of a data element is referred to as an "instance" of the data element.

17.1.1 N/A

When it appears as a value, "N/A" (shorthand for "Not Applicable") indicates that the values within the domain of this data element are not applicable in this instance.

```
INSTRUMENT_ID = "N/A"
```

For example, in the Data Set catalog object, the instrument identification associated with NAIF SPK kernels is "N/A" since these data sets have no associated instruments.

17.1.2 UNK

When it appears as a value, "UNK" (shorthand for "Unknown") indicates that the value for this data element in this instance is permanently not known. A value is applicable but none is forthcoming.

```
FILTER_NAME = "UNK"
```

In this example for a value with a character data type, the filter used for a Viking Image is not known and no archive exists that supplies this information.

```
TWIST_ANGLE = "UNK"
```

"UNK" can also be used for values that have numeric data types, as shown in this example. Here it indicates that the twist angle that applies to an image is not known and no archive exists that supplies this information.

17.1.3 NULL

When it appears as a value, "NULL" indicates that the value for this data element in this instance is temporarily unknown. A value is applicable and is forthcoming.

```
DATA_SET_RELEASE_DATE     = "NULL"
```

This example shows that a data set could be loaded into the catalog before being officially released. During the interim, the release date is not known.

17.2 Implementation recommendations for N/A, UNK, and NULL

Within information processing systems such as the PDS catalogs, the above definitions imply that three distinct values will be stored for the "figurative constants" N/A, UNK, and NULL. The PDS recommendations are as follows.

1) For character fields: The strings "N/A", "UNK", and "NULL" (see 3) can be stored as values in data elements with character data types. This includes DATE/TIME data types where UTC or other character formats are specified.

2) For numeric fields: See Table 17.1 for the values stored for data elements with numeric data types.

3) Exception: Files such as volume INDEX files that are included in archive volumes in ASCII format may use the figurative constants "N/A", "UNK", and "NULL" for both numeric and character data types. Alternatively, numeric constants representing N/A, UNK, and NULL may be defined for each column in an INDEX table, using the keywords NOT_APPLICABLE_CONSTANT, UNKNOWN_CONSTANT, and NULL_CONSTANT in the appropriate COLUMN objects.

Table 17.1: Numeric values for N/A, UNK, NULL

	Signed Integer (4 byte)	Signed Integer (2 byte)	Unsigned Integer (4 byte)	Unsigned Integer (2 byte)	Tiny Integer (1 byte - unsigned)	Real	Binary Time
N/A	-2147483648	-32768	4294967293	65533	locally defined	-1.E32	Jan. 1, 1753**
UNK	2147483647	32767	4294967294	65534	locally defined	+1.E32	Dec. 31, 9999**
NULL	Null*	Null*	null*	null*	null*	null*	null*

* The availablility of NULL as a universal value across data types in some data management systems simplifies the implementation of the figurative constant "NULL". However, if a system "null" is not available, then either a) an arbitrary value can be chosen, or b) the meanings of UNK and NULL can be combined and the token or numeric representation of UNK used.

** Sybase limits.

Chapter 18

Units of Measurement

The uniform usage of units is essential in a broadly-based catalog system, for obvious reasons. One cannot search for all the instruments covering 400 to 700 nm wavelength if some of the entries are in Angstroms and some in microns. The PDS standard shall be *Systeme Internationale d'Unites* (SI) where applicable. For example, micrometers should be used rather than microns.

The units for the data elements used in PDS data product labels and templates have been determined by the discipline scientists on a data element by data element basis. The Planetary Science Data Dictionary defines the desired units for each database element used in the system. In addition, there is a table in the PSDD that gives unit definitions.

In cases where more than one type of unit is possible for a given data element, an additional data element shall be used to identify the applicable unit. For example, the value of the element SAMPLING_PARAMETER_RESOLUTION may be given in different units, depending on the situation. Therefore, an additional element, SAMPLING_PARAMETER_UNIT, accompanies it, in order to specify the applicable unit of measure. The PDS allows exceptions to SI units when needed for consistency with previous community usage (e.g. an angle measurement in degrees instead of radians).

Both the name of the unit and the symbol are allowed as well as singular or plural form. In addition, the double asterisk (**) is used, rather than the caret (^) to indicate exponentiation, in order to comply with the preferences of the European science community.

SI Units

The following summary of SI unit information is extracted from *The International System of Units.*

Base units —As the system is currently used, there are seven fundamental SI units, termed "base units”:

QUANTITY	NAME OF UNIT	SYMBOL
length	meter	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
thermodynamic temperature	kelvin	K

amount of substance	mole	mol
luminous intensity	candela	cd

SI units are all written in lowercase style; symbols are also lowercase except for those derived from proper names. No periods are used with any of the symbols in the international system.

Derived units — In addition to the base units of the system, a host of derived units, which stem from the base units, are also employed. One class of these is formed by adding a prefix, representing a power of ten, to the base unit. For example, a kilometer is equal to 1,000 meters, and a millisecond is .001 (that is, 1/1,000) second. The prefixes in current use are as follows:

SI PREFIXES

Factor	Prefix	Symbol	Factor	Prefix	Symbol
10**18	exa	E	10**-1	deci	d
10**15	peta	P	10**-2	centi	c
10**12	tera	T	10**-3	milli	m
10**9	giga	G	10**-6	micro	
10**6	mega	M	10**-9	nano	n
10**3	kilo	k	10**-12	pico	p
10**2	hecto	h	10**-15	femto	f
10**1	deka	da	10**-18	atto	a

Although, for historical reasons, the kilogram rather than the gram was chosen as the base unit, prefixes are applied to the term gram instead of the official base unit: megagram (Mg), milligram (mg), nanogram (ng), etc.

Another class of derived units consists of powers of base units and of base units in algebraic relationships. Some of the more familiar of these are the following:

QUANTITY	NAME OF UNIT	SYMBOL
area	square meter	m**2
volume	cubic meter	m**3
density	kilogram per cubic meter	kg/m**3
velocity	meter per second	m/s
angular velocity	radian per second	rad/s
acceleration	meter per second squared	m/s**2
angular acceleration	radian per second squared	rad/s**2
kinematic viscosity	square meter per second	m**2/s
dynamic viscosity	newton-second per square meter	N*s/m**2
luminance	candela per square meter	cd/m**2
wave number	1 per meter	m**-1
activity (of a radioactive source)	1 per second	s**-1

Many derived SI units have names of their own:

QUANTITY	NAME OF UNIT	SYMBOL	EQUIVALENT
frequency	hertz		s**-1
angular acceleration	hertz	Hz	s**-1
force	newton	N	kg*m/s**2
pressure (mechanical stress)	pascal	Pa	N/m**2
work,energy,quantity of heat	joule	J	N*m
power	watt	W	J/s
quantity of electricity potential difference	coulomb	C	A*s
electromotive force	volt	V	W/A
electrical resistance	ohm	-	V/A
capacitance	farad	F	A*s/V
magnetic flux	weber	Wb	V*s
inductance	henry	H	V*s/A
magnetic flux density	tesla	T	Wb/m**2
magnetomotive force	ampere	A	
luminous flux	lumen	lm	cd*sr
illuminance	lux	lx	lm/m**2

Supplementary units are as follows:

QUANTITY	NAME OF UNIT	SYMBOL
plane angle	radian	rad
solid angle	steradian	sr

Use of figures with SI units —In the international system it is considered preferable to use only numbers between 0.1 and 1,000 in expressing the quantity of any SI unit. Thus the quantity 12,000 meters is expressed 12 km, not 12,000 m. So too, 0.003 cubic centimeters is preferably written 3 mm3, not 0.003 cm3.

Chapter 19

Volume Organization and Naming

The *Volume Organization and Naming* Standard defines the standard way of organizing data sets onto physical media and the conventions for forming volume names and identifiers. A volume is one unit of physical media such as a CD-ROM, a CD-WO, an 8mm magnetic tape, or a 9-track magnetic tape. Data sets may reside on one or more volumes and multiple data sets may also be stored on a single volume. Volumes are grouped into Volume Sets.

Each volume has a directory structure which contains subdirectories and files. Both random access (CD-ROM) and sequential access (magnetic tape) media are supported. A PDS volume on sequential access media has a "virtual" directory structure defined in the volume object included on the volume in the file VOLDESC.CAT. The virtual directory structure may be used to recreate the volume directory structure when the files are moved to random access media.

PDS recommends that archive volumes be based on a single version of the PDS Standards. Software tools that work with one version of the standard may not work with all versions.

19.1 Volume Set Types

Data may be organized into one of four types of archive volumes. The distinguishing characteristics between the volume types are the number of data sets on each volume and the number of volumes required to capture all the data. The directory organization of the volumes and the required files varies slightly depending on the volume type. Figures 19.1 through 19.5 depict the various volume directory structure options. The four volume types are described below.

(1) One data set on one volume - this is the basic volume organization consisting of the required ROOT directory, INDEX, and data subdirectories and the seven optional subdirectories: DOCUMENT, CATALOG, LABEL, GAZETTER (not shown in the figures), SOFTWARE, CALIB, and GEOMETRY. See Figure 19.1.
Note that CALIB and GEOMETRY are only recommended directory names, other appropriate names may be substituted.

(2) One data set on many volumes - this type includes both an index for the volume and a cumulative index for the volume set (up to the given volume number, not the entire set) in the INDEX subdirectory. See Figure 19.2.

(3a) Many data sets on one volume (one logical volume) - this type of volume requires additional file naming conventions to distinguish similar files for different data sets. In addition, the DATA subdirectories are organized by data set (or equivalent, e.g. instrument) at the first level below the ROOT directory. See Figure 19.3.

(3b) Many data sets on one volume (many logical volumes) - this volume organization is designed to accommodate many small data sets that have distinct documentation, indexing and other ancillary information that are more logically packaged together below the root directory of the volume. See Figure 19.4. Directories common to all logical volumes (e.g. SOFTWARE) may also be supplied, provided there are no pointer references to any files within a common directory.

(4) Many data sets on many volumes - this type requires additional file naming conventions, cumulative indices, and a first level subdirectory organization by data set. See Figure 19.5.

NOTE: It is permissible to have one or more data volumes with an ancillary volume containing the DOCUMENT, CATALOG, GAZETTER, SOFTWARE, CALIB, and GEOMETRY directories. If this is done, PDS requires that all include files be present on each data disk. PDS prefers that ancillary files be archived on the same volumes as the data wherever possible. This makes data easier to access for the science users. The contents and organization of the directories of all the volume types are described in this chapter.

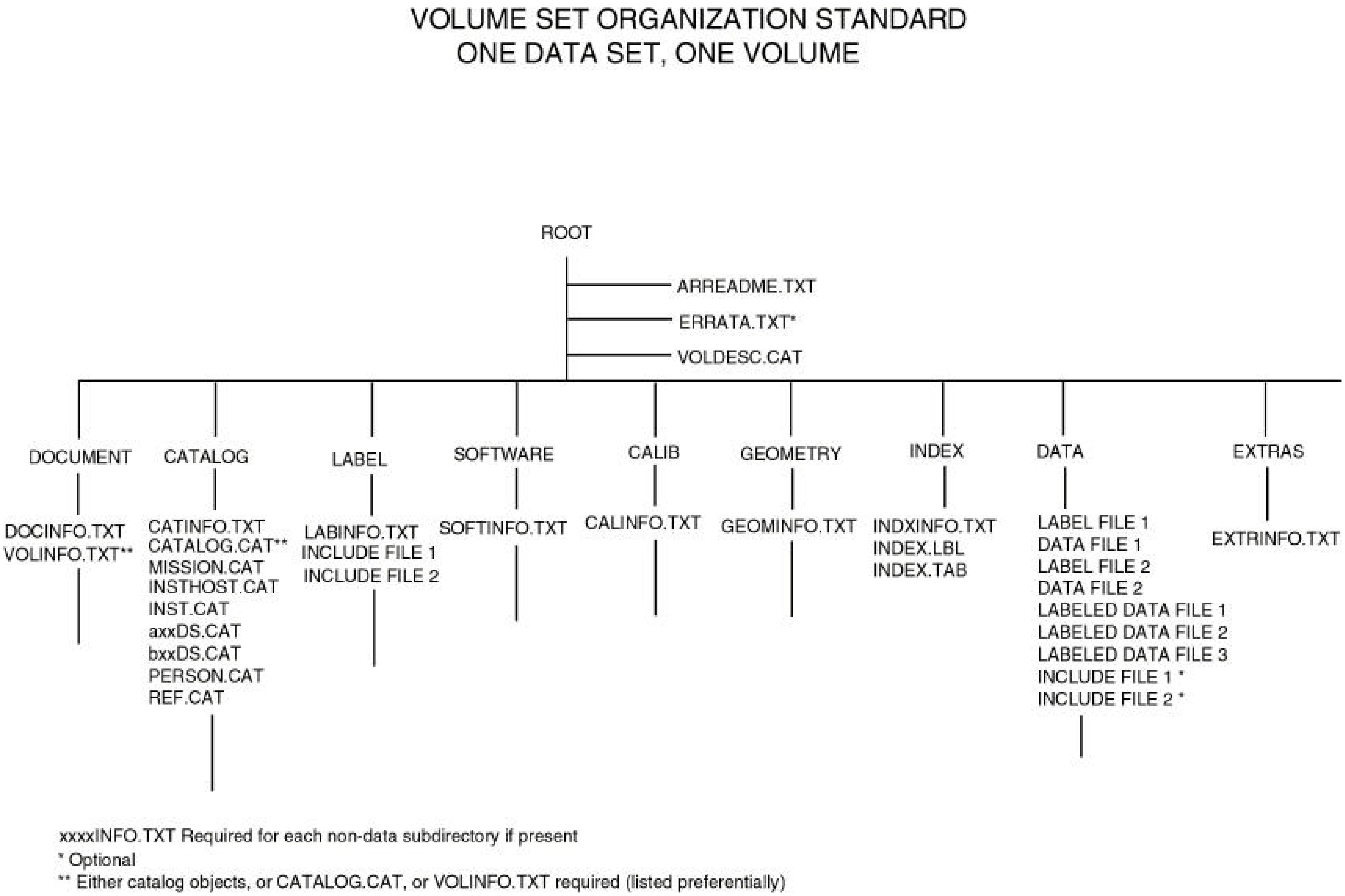
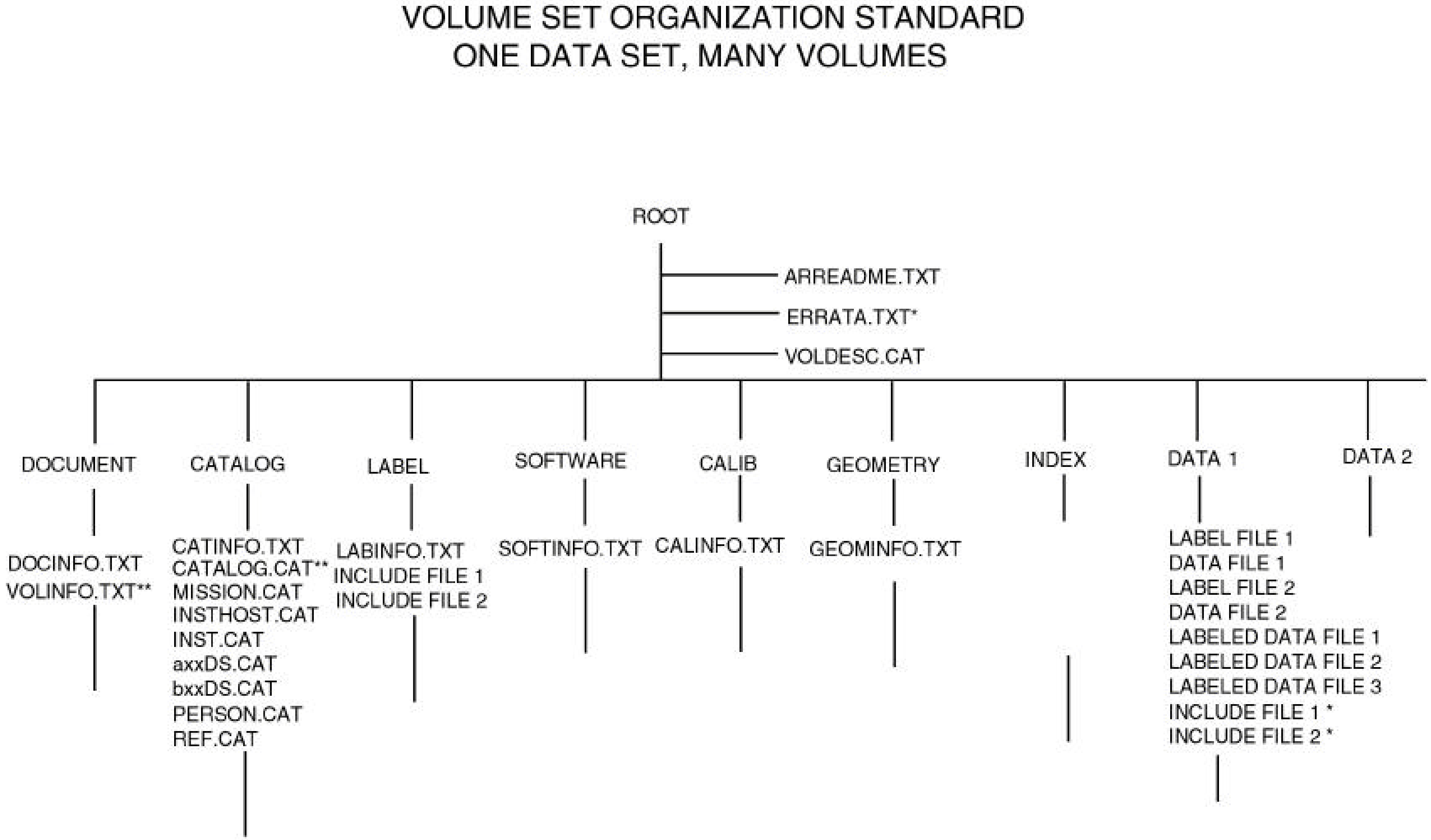
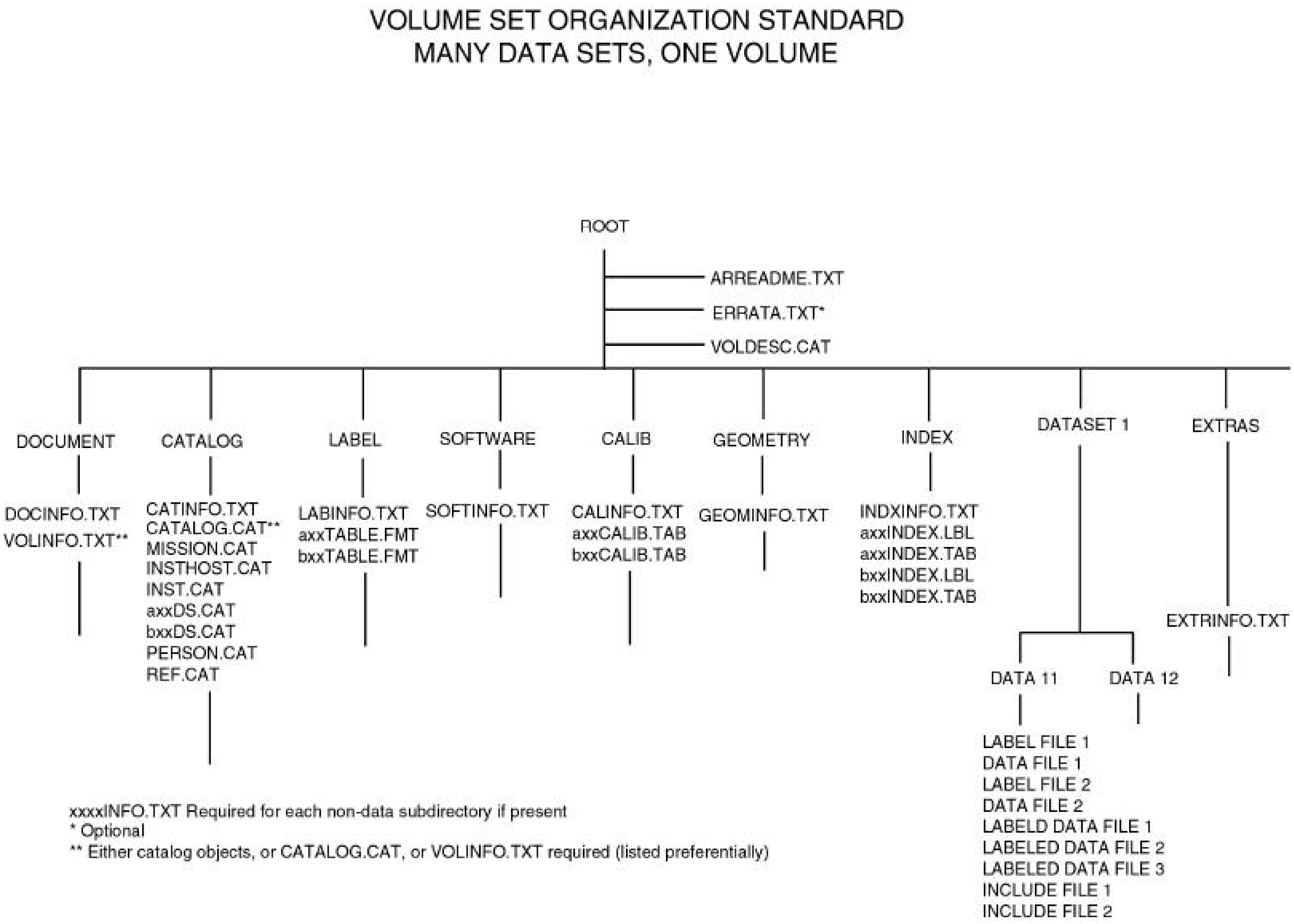


Figure 19.1 Volume Set Organization Standard - One Data Set, One Volume



xxxxINFO.TXT Required for each non-data subdirectory if present
* Optional
** Either catalog objects, or CATALOG.CAT, or VOLINFO.TXT required (listed preferentially)

Figure 19.2 Volume Set Organization Standard - One Data Set, Many Volumes



xxxxINFO.TXT Required for each non-data subdirectory if present
* Optional
** Either catalog objects, or CATALOG.CAT, or VOLINFO.TXT required (listed preferentially)

Figure 19.3 Volume Set Organization Standard - Many Data Sets, One Volume

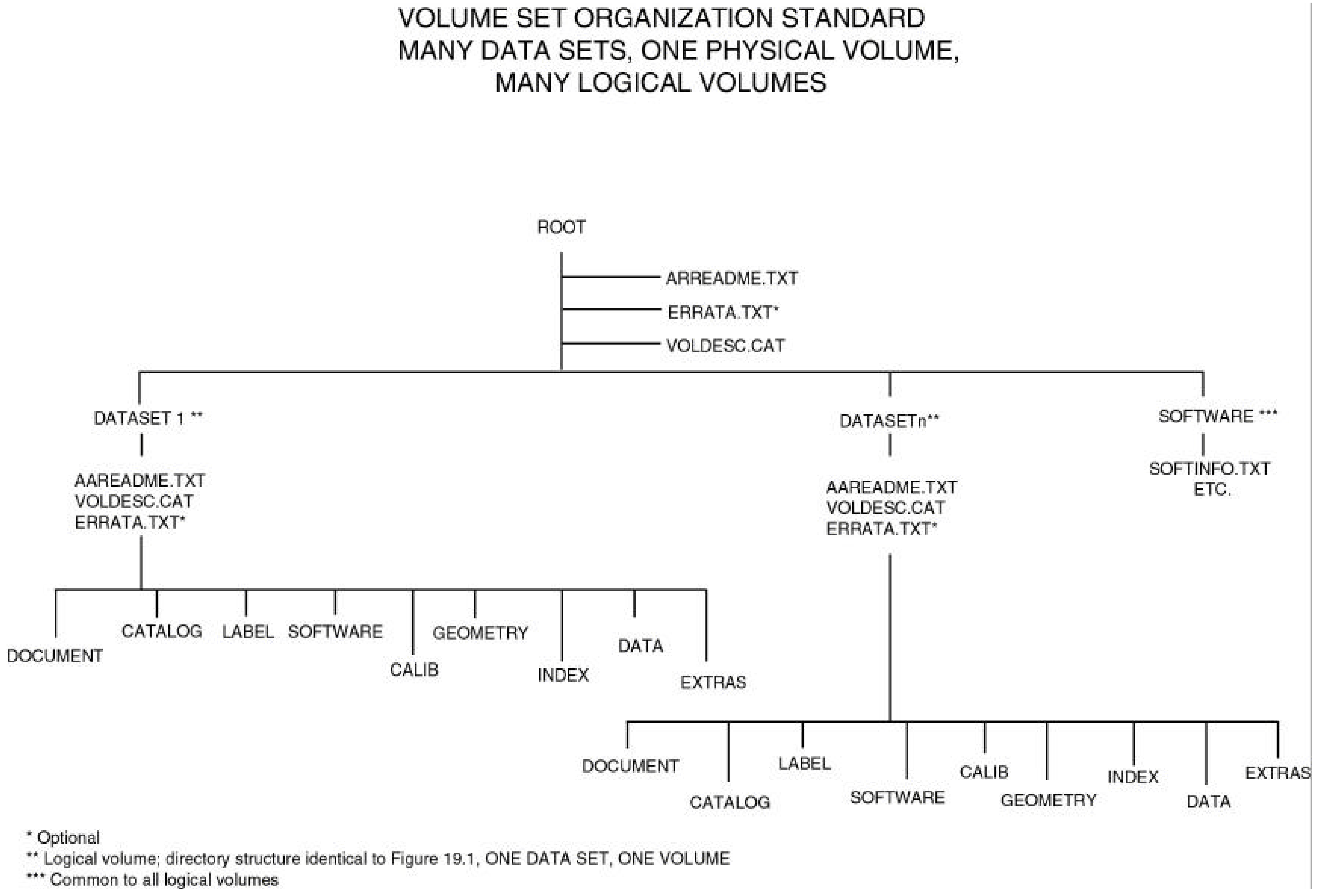
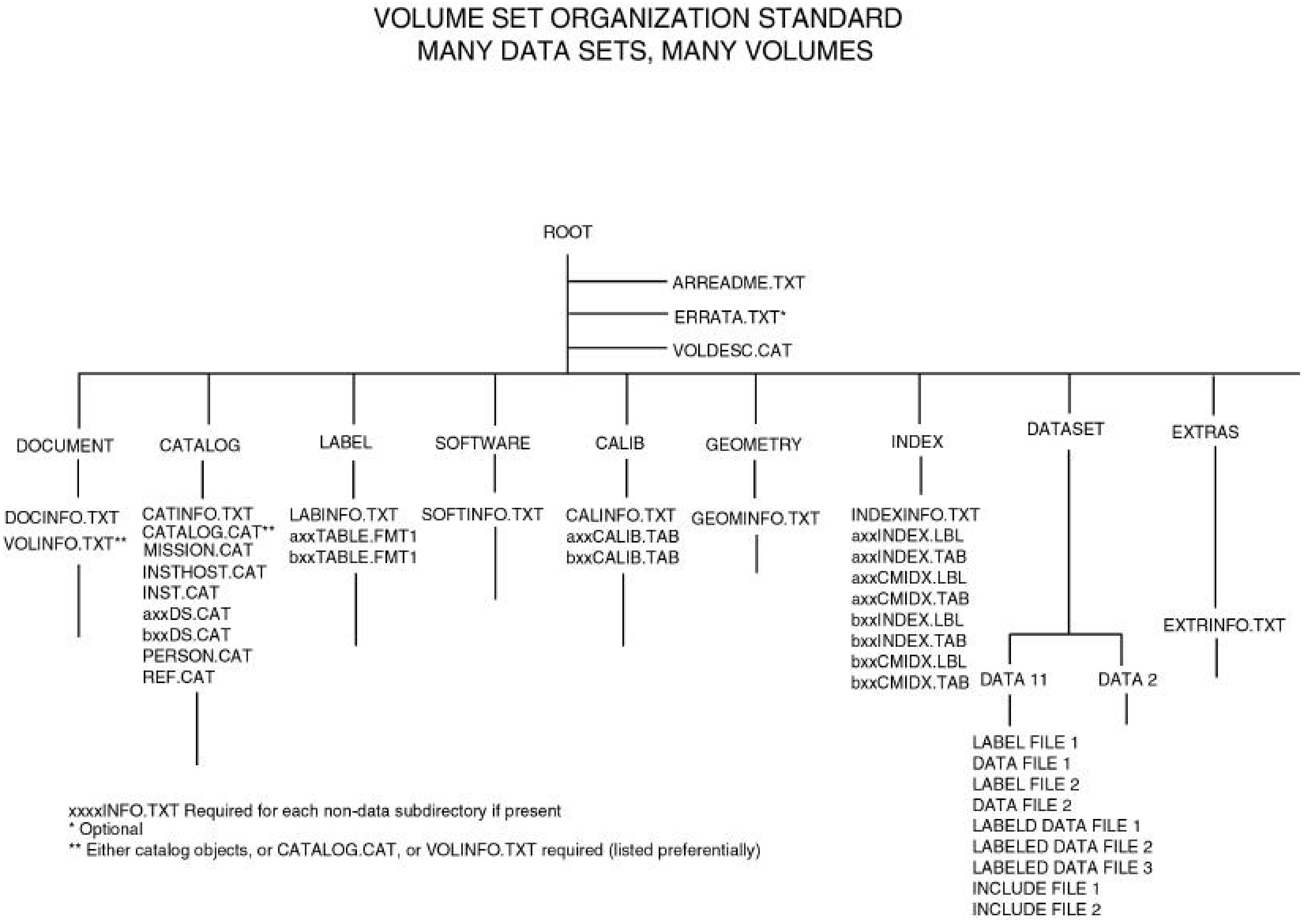


Figure 19.4 Volume Set Organization Standard - Many Data Sets, One Physical Volume, Many Logical Volumes



xxxxINFO.TXT Required for each non-data subdirectory if present
* Optional
** Either catalog objects, or CATALOG.CAT, or VOLINFO.TXT required (listed preferentially)

Figure 19.5 Volume Set Organization Standard - Many Data Sets, Many Volumes

19.2 Volume Organization Guidelines

PDS recommends that directory structures be simple, path names short, and directory and file names be constructed in a logical manner. It is recommended that the number of files per subdirectory should ideally be a screenful, allowing users to browse through file names using the directory command. Some externally developed software cannot handle subdirectories with more than 255 files, so it is recommended that this number not be exceeded. PDS also recommends that there be no empty subdirectories (as a convenience to users).

19.3 Description of Directory Contents and Organization

ROOT Directory -- Required
Top level directory of a physical or logical volume. The ROOT directory (of a physical or logical volume) contains the following required and optional files and subdirectories.

AAREADME.TXT -- Required
Contains an overview of the contents of the volume (physical or logical volume) and its organization, general instructions for using the volume and its contents, and provides contact information. Its name has been chosen so that it will be listed first in an alphabetical directory listing. See Appendix D for an outline and example of an AAREADME.TXT file.

ERRATA.TXT -- Optional
Contains textual information describing errors and/or anomalies found in the current volume as well as errors and/or anomalies found in previous volumes of a volume set. If known errors exist on a volume they shall be documented in this file.

VOLDESC.CAT -- Required
Contains the VOLUME Object which gives a high-level description of the contents of the volume.

VOLDESC.SFD -- Optional
Contains the SFDU Reference Object structure which aggregates the separate file contents of the volume into an SFDU. The Reference Object is expressed in PVL. This file should only be considered for use if the data products are packaged as SFDUs. Note: the ".SFD" file extension is a reserved file extension in the CCSDS SFDU standard indicating the file contains a valid SFDU. Note that this file is identified here for backward compatability with previous versions of the PDS standards and is not to be used in current archive products

DOCUMENT Subdirectory -- Optional
Contains all the textual material that describes the mission, spacecraft, instrument, and data set. This can include references to science papers, or the actual papers.

DOCINFO.TXT -- Required

Contains a textual description of the contents of the DOCUMENT subdirectory.

VOLINFO.TXT -- Optional
Contains a textual description of the contents of the volume. It is an optional file, however, either one or both of the VOLINFO.TXT or the data set catalog objects in the CATALOG subdirectory shall be included on the volume (see the CATALOG subdirectory).

CATALOG Subdirectory -- Optional
Contains all the completed catalog objects for the mission(s), instrument host(s), instrument(s), and data set(s) for the archive volume. This is an optional directory (i.e., a complete set of catalog objects do not have to be included on the archive volume); if and only if, a VOLINFO.TXT file is included in the DOCUMENT directory of the volume. If a complete set of catalog objects are provided then the VOLINFO.TXT is not required, and vice versa. The VOLINFO.TXT file is the textual equivalent of the catalog objects (i.e., all of the information required to be present in the catalog objects shall be present in the VOLINFO.TXT file).

The primary difference between catalog objects and the VOLINFO.TXT file is that the catalog objects use an "OBJECT / END_OBJECT" and "KEYWORD = VALUE" format and each catalog object is an independent file. The VOLINFO.TXT is a single file which uses a text format.

Note that for logical volumes, these must be below the logical volume root, if present.

CATINFO.TXT -- Required
Contains a textual description of the contents of the CATALOG subdirectory.

CATALOG.CAT -- Required
Contains the entire set of high-level descriptive information about a data set (this includes separate descriptions for each mission, each instrument host, each instrument, and each data set; as well as, reference and personnel information), expressed in PDS objects which makes the file suitable for loading into a catalog.

PDS Preferred Method for Supplying Catalog Objects

Individual catalog objects may also be packaged into separate files. This is the preferred method for supplying catalog objects as each catalog object must be ingested into the PDS catalog independent of the other objects. If a CATALOG.CAT file is supplied, the CN data engineer must disassemble the single file into multiple files / catalog templates. Each file corresponds to a catalog template and each catalog template corresponds to a single catalog object. The data engineer then must validate each file, and format each description field in accordance with the prescribed headings and sub-headings for each catalog template.

For example, in Figure 19.5, the files axxxxxDS.CAT and bxxxxxDS.CAT represent two separate files each containing data set catalog objects (descriptive information about the data set) for data sets a and b respectively. See the *File Specification and Naming* chapter in this document for the file naming rules. See also Appendix A for the required contents of the catalog object.

Note that the axx- and bxx- prefixes in the sample names are neither required nor recommended. Data producers may use them to distinguish two or more files (by data set, instrument, or other criterion). The data producer should replace the generic prefixes shown here by a suitable mnemonic acronym.

LABEL Subdirectory -- Optional
Contains additional PDS labels and/or include files (meta data or descriptive information) which were not packaged with the data products or in the data subdirectories.
Note that if a logical volume organization is used, the LABEL subdirectory, if present, must reside below the logical volume ROOT, since pointer references to files within a common directory are not allowed.

LABINFO.TXT -- Required
Contains a textual description of the contents of the LABEL subdirectory.

Include Files -- Required
Files pointed to in a PDS label that contain additional meta data or descriptive information. Only files of type LBL, TXT, or FMT shall be included in the LABEL subdirectory. In the figures, the files axxINCLUDE $FILE_1$, bxxINCLUDE $FILE_1$ and INCLUDE $FILE_1$ represent sample files of the above types. The axx and bxx prefixes indicate that the include files for different data sets (a and b) may be combined in the same LABEL subdirectory.

Note that the axx- and bxx- prefixes in the sample names are neither required nor recommended. Data producers may use them to distinguish two or more files (by data set, instrument, or other criterion). The data producer should replace the generic prefixes shown here by a suitable mnemonic acronym.

GAZETTER Subdirectory -- Optional
Contains detailed information about all the named features on a target body associated with the data sets on the volumes. The features are those the International Astronomical Union (IAU) has named and approved.

GAZINFO.TXT -- Required
Contains a textual description of the contents of the GAZETTER subdirectory.

GAZETTER.TXT -- Required
Contains a textual description of the structure and contents of the gazetteer table.

GAZETTER.LBL -- Required
Contains the PDS label identifying and giving a formal description of the structure of the gazetteer table.

GAZETTER.TAB -- Required
Contains the gazetteer table.

SOFTWARE Subdirectory -- Optional
Contains the software libraries, utilities, or application programs to access/process the data objects. It may also include algorithms. Currently only public domain software can be included on PDS archive volumes.

The following SOFTWARE subdirectory structure is the recommended platform-based model. An alternative model for the SOFTWARE subdirectory structure is application-based (e.g. directory names are based on the application such as DISPLAY). See Appendix D SOFTINFO.TXT example for the subdirectory structure used for Clementine. See Appendix E for the subdirectory structure of the NAIF Toolkit for a single platform.

SOFTINFO.TXT -- Required
Contains a textual description of the contents of the SOFTWARE subdirectory.
For an outline and example, see Appendix D.

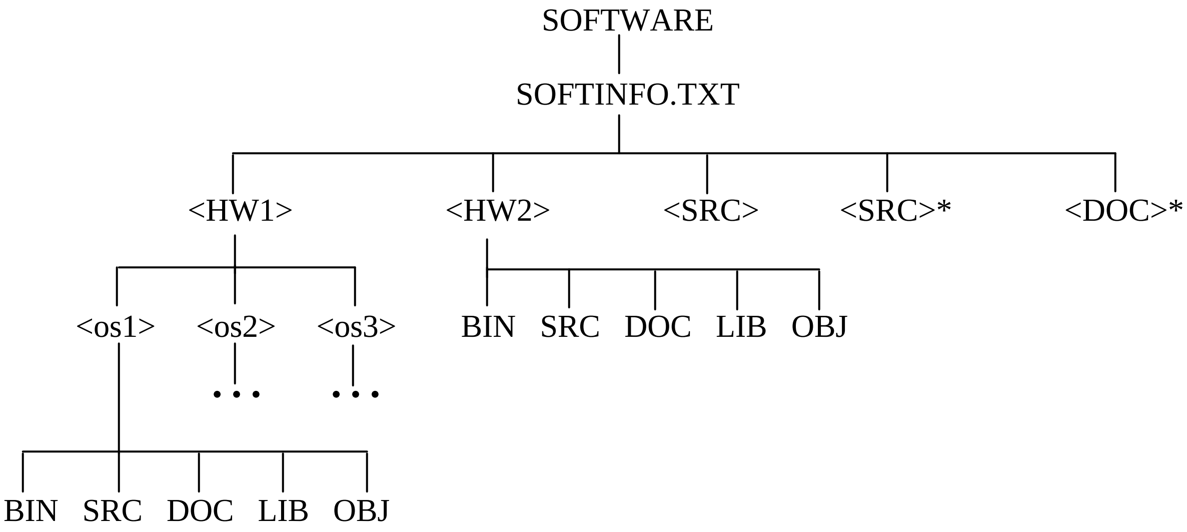
SRC Subdirectory -- Optional
There can be a global SRC directory under the SOFTWARE directory if there is source code applicable to all platforms. For example, application programming languages such as IDL are relatively platform independent and would be placed in a global SRC directory. Note in example below, there is both a global source directory as well as source directories at the lower levels.

DOC Subdirectory -- Optional
A global DOC directory under the SOFTWARE directory would contain documentation for the source code in the global SRC directory.

LIB Subdirectory -- Optional
A global LIB directory under the SOFTWARE directory would contain libraries applicable to all platforms.

Hardware Platform and Operating System/Environment Subdirectories -- Optional (not present if only global source code provided)

1. The hardware platform and the operating system/environment must be explicitly stated. If there is more than one operating system/environment (os/env) supported then they must be subdirectories under the hardware directories. If there is only one, then that subdirectory can be promoted to the hardware directory level (via naming conventions). In the example below, since only one os/env is supported on hardware 2, the name of the hardware subdirectory also contains the os/env name.



2. The next level of directories are BIN, SRC, DOC, LIB and OBJ. If any are not applicable, they should be left off (i.e. no empty directories).

*info.txt files under SOFTWARE subdirectories are optional (e.g. PCINFO.TXT, MACINFO.TXT, VAXINFO.TXT, SUNINFO.TXT, etc.).

3. Examples of subdirectory names for the two cases where there are single or multiple operating system/environments are listed below. This list is not meant to be a complete list, it will be updated on an as-needed basis.

Multiple	Single
PC	
DOS	PCDOS
WIN	PCWIN
WINNT	PCWINNT
OS2	PCOS2
MAC	
SYS7	MACSYS7
AUX	MACAUX
SUN	
SUNOS	SUNOS
SOLAR	SUNSOLAR

VAX	
VMS	VAXVMS
ULTRX	VAXULTRX
SGI	
IRX4	SGIIRX4
IRX5	SGIIRX5

CALIBration Subdirectory -- Optional
Contains the calibration files used in the processing of the raw data or needed to use the data products on the volume.
Note that CALIB is only a recommended directory name, another appropriate name may be used.

CALINFO.TXT -- Required
Contains a textual description of the contents of the CALIB subdirectory.

Calibration Files -- Required
In the figures, the files axxCALIB.TAB and bxxCALIB.TAB represent sample files. The axx and bxx prefixes indicate that the calibration files for different data sets (a and b) may be combined in the same CALIB subdirectory.

Note that the axx- and bxx- prefixes in the sample names are neither required nor recommended. Data producers may use them to distinguish two or more files (by data set, instrument, or other criterion). The data producer should replace the generic prefixes shown here by a suitable mnemonic acronym.

GEOMETRY Subdirectory -- Optional
Contains the relevant files (e.g., SEDRs, SPICE kernels) needed to describe the observation geometry.
Note that GEOMETRY is only a recommended directory name, another appropriate name may be used.

GEOMINFO.TXT -- Required
Contains a textual description of the contents of the GEOMETRY subdirectory.

INDEX Subdirectory -- Required (exception noted below)
Contains the indices for the data products in the data set(s) on the volume.

Exception note: If the logical volume organization is used, there will generally be no INDEX subdirectory at the ROOT of the physical volume. Instead there will be individual INDEX subdirectories at the ROOT of each logical volume.

INDXINFO.TXT -- Required
Contains a textual description of the contents of the INDEX subdirectory. This description should include at least:

1) A description of the structure and contents of each index table in this subdirectory.

2) Usage notes

For an example of the INDXINFO.TXT file, see Appendix D, Section D.2.

INDEX.LBL -- Required (exception noted below)
For all volumes, this file contains the PDS label for the volume index (INDEX.TAB). The INDEX_TABLE specific object should be used to identify and describe the structure (columns) of the index table. See Appendix A.
Although INDEX.LBL is the preferred name for this file, the name axxINDEX.LBL may also be used (with axx replaced by an appropriate mnemonic).

Exception note: PDS recommends the use of detached labels for index tables. If an attached label is used, this file is superfluous (i.e., not needed).

INDEX.TAB -- Required
For all volumes, this file contains the volume index in tabular format. Normally only data files are included in an index table. In some cases, however, ancillary files may be included.
Although INDEX.TAB is the preferred name for this file, the name axxINDEX.TAB may also be used (with axx replaced by an appropriate mnemonic).

Note that the axx- and bxx- prefixes in the sample names are neither required nor recommended. Data producers may use them to distinguish two or more files (by data set, instrument, or other criterion). The data producer should replace the generic prefixes shown here by a suitable mnemonic acronym.

CUMINDEX.LBL -- Recommended for multi-volume sets
For multi-volume sets, this file contains the PDS label for the cumulative volume set index (CUMINDEX.TAB). The INDEX_TABLE specific object should be used to identify and describe the structure (columns) of the cumulative volume set index table. See Appendix A.
Although CUMINDEX.LBL is the preferred name for this file, the name axxCMIDX.LBL may also be used (with axx replaced by an appropriate mnemonic).
PDS recommends the use of detached labels for index tables. If an attached label is used, this file is not needed.

CUMINDEX.TAB --Recommended for multi-volume sets
For multi-volume sets, this file contains the cumulative volume set index in a tabular format. Normally only data files are included in a cumulative index table. In some cases, however, ancillary files may be included.
Although CUMINDEX.TAB is the preferred name for this file, the name axxCMIDX.TAB may also be used (with axx replaced by an appropriate mnemonic).

EXTRAS Subdirectory -- Optional
The EXTRAS directory is the designated area for housing additional elements provided by data

producers beyond the scope of PDS compliance requirements of a data set. Examples include HTML-based disk navigators, educational and public interest aids, and other useful but nonessential items. The PDS has no restrictions on the contents and organization of this subdirectory other than conformance to ISO-9660/UDF standards.

EXTRINFO.TXT -- Required
Contains a textual description of the contents and organization of the EXTRAS subdirectory. This description should include at least:

1) A description of the structure and contents of each file in this subdirectory.

2) Usage notes

Data Subdirectories -- Required (exception noted below)
Contain the data product files. These subdirectories are organized and named according to the *Directory Types and Naming* chapter in this document. Subdirectories may be nested up to eight levels deep on a physical volume. Data products may be packaged with their PDS labels attached, where the label and the data object(s) are contained in a LABELED DATA FILE, or with PDS labels detached, where the PDS label is contained in a LABEL FILE and the data object(s) in a DATA FILE.

Data File -- Contains a data object which is a grouping of data resulting from a scientific observation such as an image or table, representing the measured instrument parameters. The associated PDS label is contained in a LABEL FILE.

Label File -- Contains a detached PDS label expressed in the Object Definition Language that identifies, describes, and defines the structure of the data objects. The associated data objects are contained in a DATA FILE. The LABEL FILE shall have the same basename as the associated DATA FILE and the extension of ".LBL".

Labeled Data File -- Contains data object(s) and associated PDS label.

Exception note: Data subdirectories are not present at the ROOT level of a physical volume when logical volumes are used. Instead, they are nested below the ROOT of the logical volume.

19.4 Volume Naming

The Volume name provides the name of a data volume. Volume names shall be at most 60 characters in length and are in upper case. They should describe the contents of the volume in terms that a human user can understand. Most computer systems and software use the volume ID, not the volume set name or volume name, when processing media volumes. The volume set name or volume name are therefore more important to a human user than to a machine.

In most cases the volume name is more specific than the volume set name. For example, the volume name for the first volume in the VOYAGER IMAGES OF URANUS volume set is:

“VOLUME 1: COMPRESSED IMAGES 24476.54 - 26439.58"

19.4.1 Volume ID

Many types of media and the machines that read media volumes place a limit on the length of the volume ID. Therefore, although the complete volume set ID should be placed on the outside label of the volume, a shorter version is actually used when the volume is recorded. PDS has adopted a limit of 9 characters for these terse volume identifiers. This terse identifier shall consist of the last two components of the volume set ID, with the "X" wildcard values replaced by the sequence number associated with the particular volume (see the *Volume Set ID* Standard below). This ID must always be unique for PDS data volumes. Note that the ID must be in upper case.

EXAMPLES:

VG_0002 (for volume 2 of the Voyager set)
MG_0001 (for the first volume of the Magellan set)
VGRS_0001 (for a potential Voyager Radio Science collection)

If a volume is redone because of errors in the initial production the volume ID should remain the same, and the VOLUME_VERSION_ID should be incremented. This parameter is contained in the VOLDESC.CAT file on the volume, and the version ID should also be placed on the external volume label as “Version n” where n indicates the revision number. This indicates that the original volume should be replaced with the new version. If a volume is redone because the data have been enhanced it should be given a new volume ID, not a new version number.

19.5 Volume Set Naming

The Volume Set Name provides the full, formal name of a group of data volumes containing a data set or a collection of related data sets. Volume set names shall be at most 60 characters in length and must be in upper case. Volume sets are normally considered as a single orderable entity.
For example, the volume series MISSION TO VENUS consists of the following volume sets:

MAGELLAN: THE MOSAIC IMAGE DATA RECORD

MAGELLAN: THE ALTIMETRY AND RADIOMETRY DATA RECORD

MAGELLAN: THE GLOBAL ALTIMETRY AND RADIOMETRY DATA RECORD

PRE-MAGELLAN RADAR AND GRAVITY DATA SET COLLECTION

In certain cases, the volume set name can be the same as the volume name, such as when the volume set consists of only one volume.
Note that in VAX computer usage a volume set has very special attributes, and that all volumes of a volume set must be on line for proper access. There are no plans within PDS to produce volume sets following the VAX definition. Instead the VOLUME_SET_NAME and

VOLUME_SET_ID are used to group related data and to provide additional specificity in a volume name in case volumes produced by different organizations have the same volume IDs.

19.5.1 Volume Set ID

The volume set ID identifies a data volume or a set of volumes. Volume sets are normally considered as a single orderable entity. Volume set IDs shall be at most 60 characters in length, must be in upper case, and are formed of the following fields, separated by underscores:

1. The country of origin (abbreviated).
2. The government branch.
3. The discipline within the branch that is producing the volumes.
4. A campaign, mission or spacecraft identifier (2 characters) followed by an optional 2 character instrument or product identifier.
5. A 4 digit sequence identifier. The first digit or digits may be used to represent the volume set and the trailing “X”s are wildcards that represent the range of volumes in the set. Up to 4 "X"s are allowed.

EXAMPLE

USA_NASA_PDS_GO_10XX could be the Volume set ID for the Galileo EDR volume set,since there are less than 100 volumes (since the XX placeholder accommodates the range 01 - 99 only). Note that the volume IDs for volumes in the set would then be GO_1001, GO_1002, etc.

NOTE: Prior to version 3.2, the 4-digit sequence identifier (item 5 above) did not include the “X”s. currently used as wildcards. Instead, the last digits represented the volume. For example, on Magellan, a volume_set_ID "USA_NASA_JPL_MG_0001" was used ONLY for the volume with volume_ID of "MG_0001". Subsequent volumes in the same set had volume_set_IDs that differed in the final field.

If a set of volumes was to be distributed as one logical unit, the volume set ID included the range of volume IDs.

EXAMPLE

USA_NASA_PDS_VG_0001_TO_VG_0003 for the three volumes that comprise the Voyager Uranus volume set.

19.6 Logical Volume Naming

Logical volumes will retain the volume and volume set naming used at the physical volume level. For further information, see Appendix A, Volume Object.

19.7 Exceptions to This Standard

In some rare cases, machine or software restrictions may exist on volume IDs. Also, volumes made in the past may have IDs which do not meet this standard and there may be compelling reasons for keeping the same volume ID when making a new copy of the data. All new data sets, however, should use this standard.

Chapter 20

Zip Compression

The PDS standards support two different approaches to data compression.

In one case, a data object contains numbers that have been encoded using one of several supported methods (e.g., "Huffman first difference"). In this approach, the label describes the compressed data and the ENCODING_TYPE keyword indicates how the data object is to be decompressed by the user. PDS standards only support this approach to compression for IMAGE objects.

In the alternative approach, a standard compression method called "Zip" is used. In this case, an entire data file is compressed rather than a particular data object. The user is expected to apply the "Unzip" utility to decompress the file, and the label then describes the decompressed data directly.

This chapter describes PDS standards for archiving data using Zip compression. For more information on compression of individual IMAGE objects, see Sect. A.19.

In general, the archiving of data in a compressed format should be used sparingly, because although it reduces the number of physical volumes, it makes the data more difficult for users to interpret. PDS recommends that data compression should only be used in limited situations, such as to compress very large and infrequently used data or to archive processed data where the source product is readily available in a non-compressed PDS archive.

20.1 Info-Zip Software

PDS has adopted the *Zip* and *UnZip* software packages, as developed by the *Info-Zip Consortium.* A thorough description of the software packages and the Info-Zip work group can be found at:

http://www.cdrom.com/pub/infozip

This same information is available on line from PDS at:

http://pds.jpl.nasa.gov

The primary reasons for adopting the *Info-Zip* software packages include:

- *Info-Zip*, a diverse Internet-based workgroup of about 20 primary authors and over one hundred beta-testers, provides free, portable, high-quality versions of the Zip and UnZip utilities.

- *Info-Zip* has defined a lossless compressed data format that is independent of CPU type, operating system, file system, and character set. The Info-Zip utilities can be implemented readily in a manner not covered by patents, and hence can be practised and distributed freely.

- The Zip and UnZip utilities are free, as is the source code. The Zip utility is useful for packaging a set of files for distribution, for archiving files, and for saving disk space by compressing files or directories. Zip puts one or more compressed files into a single ZIP archive, along with information about the files (name, path, date, time of last modification, protection, and check information to verify file integrity). An entire directory structure can be packed into a ZIP archive with a single command. Zip has one compression method (deflation) and can also store files without compression. Zip automatically chooses the better of the two for each file.

- Compression ratios of 2:1 to 3:1 are common for text files.

- The UnZip utility is an extraction utility for archives compressed in .zip format (also called "zipfiles"). UnZip will list, test, or extract files from a .zip archive. The default behavior (with no options) is to extract into the current directory (and subdirectories below it) all files contained within the specified zipfile.

20.2 Zip File Labels

When archiving data in Zip format, two files need to be considered: (1) the zipfile itself, and (2) the data file that one obtains when one decompresses the zipfile. PDS strongly recommends that the two files have the same name but different extensions: ".ZIP" for the zipfile and a more descriptive extension (e.g. ".DAT" or ".IMG") for the unzipped file. The ".ZIP" file extension is reserved exclusively for zip-compressed files within the PDS.

PDS does not recommend the practice of compressing multiple data files into a single zipfile. This will minimize the potential confusion to a user not able to locate a desired file because it was hidden inside a differently-named zipfile. It also reduces the risk associated with compressing data and is akin to "not putting all the eggs into one basket". The only exception to this rule is that multiple files in the same directory that have the same name but different extensions can be archived in the same zipfile. For example, if file ABC.IMG contains an image and file ABC.TAB contains a table of additional information relevant to that image, then both files can be archived in the file ABC.ZIP. (As described below, PDS detached label files are also included in zipfiles.)

Like all PDS data files, both the zipped and the unzipped data files require labels. These files must be described by a single, detached PDS label file, via the combined-detached label approach (see Sect. 5.2.2). Attached labels are not permitted for Zip-compressed data, because the user must be able to examine the label before deciding whether or not to decompress the file. In a combined-detached label, each individual file is described within a FILE object. Here is the general framework:

```
PDS_VERSION_ID                                          = PDS3
DATA_SET_ID                                             = ...
PRODUCT_ID                                              = ...
          (other parameters relevant to both Zipped and Unzipped files)

OBJECT                                                  = COMPRESSED_FILE
          (parameters describing the compressed file)
END_OBJECT                                              = COMPRESSED_FILE

OBJECT                                                  = UNCOMPRESSED_FILE
          (parameters describing the first uncompressed file)
END_OBJECT                                              = UNCOMPRESSED_FILE

OBJECT                                                  = UNCOMPRESSED_FILE
          (parameters describing the a second uncompressed file, if present)
END_OBJECT                                              = UNCOMPRESSED_FILE
END
```

The first FILE object, the COMPRESSED_FILE, refers to the zipped file; additional FILE objects, called UNCOMPRESSED_FILEs, refer to the decompressed data file(s) that the user will obtain by unzipping the first.

The zipfile is described via a "minimal label" (Section 5.2.3). The following keywords are required:

```
FILE_NAME                    = name of the zipfile
RECORD_TYPE                  = UNDEFINED
ENCODING_TYPE                = ZIP
INTERCHANGE_FORMAT           = BINARY
UNCOMPRESSED_FILE_NAME       = a list of the names of all the files archived in the zipfile
REQUIRED_STORAGE_BYTES       = approximate total number of bytes in the data files
DESCRIPTION                  = a brief description of the zipfile format
```

Typically, the DESCRIPTION is given as a pointer to a file "ZIPINFO.TXT" found in the DOCUMENT directory on the same volume.

The subsequent UNCOMPRESSED_FILE object(s) contain complete descriptions of the data files obtained by unzipping the zipfile.

20.3 Packaging Zip Archives on Volumes

By providing the combined-detached label as presented above, a PDS volume containing zipfiles would conform to all established PDS standards, *provided both the zipfile and its constituent data files were archived*. The unique feature of a Zip-compressed PDS archive volume is that only the zipfiles appear; the UNCOMPRESSED_FILE objects described by the labels are not present on the volume, but can be obtained by unzipping the zipfiles provided.

In addition to archiving the data files in a zipfile, PDS requires that the corresponding label file also be included in the zipfile. It is recommended that any .FMT files referenced by ^STRUCTURE keywords in the label also be included. The reason is that this guarantees that, when a user transfers a zipfile from a disk and unzips it, the required label information will also be present in the same directory. Thus, the identical label is duplicated both inside and outside the zipfile.

Note: These additional .LBL and .FMT files do not need to be described by UNCOMPRESSED_FILE objects in the label, because PDS label and format files never require labels. Furthermore, the sizes of these files do not need to be included in the value of the REQUIRED_STORAGE_BYTES keyword. However, the names of these files do need to be included in the list of UNCOMPRESSED_FILE_NAME values.

20.4 Label Example

The following is an example of a PDS label for a Zip-compressed data file.

```
PDS_VERSION_ID                       = PDS3
DATA_SET_ID                          = "HST-S-WFPC2-4-RPX-V1.0"
SOURCE_FILE_NAME                     = "U2ON0101T.SHF"
PRODUCT_TYPE                         = OBSERVATION_HEADER
PRODUCT_CREATION_TIME                = 1998-01-31T12:00:00

OBJECT                               = COMPRESSED_FILE
  FILE_NAME                          = "0101_SHF.ZIP"
  RECORD_TYPE                        = UNDEFINED
  ENCODING_TYPE                      = ZIP
  INTERCHANGE_FORMAT                 = BINARY
  UNCOMPRESSED_FILE_NAME             = { "0101_SHF.DAT", "0101_SHF.LBL"}
  REQUIRED_STORAGE_BYTES             =   34560
  ^DESCRIPTION                       = "ZIPINFO.TXT"
END_OBJECT                           = COMPRESSED_FILE

OBJECT                               = UNCOMPRESSED_FILE
  FILE_NAME                          = "0101_SHF.DAT"
  RECORD_TYPE                        = FIXED_LENGTH
  RECORD_BYTES                       = 2880
  FILE_RECORDS                       = 12
  ^FITS_HEADER                       = ("0101_SHF.DAT",     1 <BYTES>)
  ^HEADER_TABLE                      = ("0101_SHF.DAT", 25921 <BYTES>)

  OBJECT                             = FITS_HEADER
    HEADER_TYPE                      = FITS
    INTERCHANGE_FORMAT               = ASCII
    RECORDS                          = 7
    BYTES                            = 20160
    ^DESCRIPTION                     = "FITS.TXT"
  END_OBJECT                         = FITS_HEADER

 OBJECT                              = HEADER_TABLE
    NAME                             = HEADER_PACKET
    INTERCHANGE_FORMAT               = BINARY
    ROWS                             = 965
  COLUMNS                              = 1

  ROW_BYTES                            = 2
  DESCRIPTION                          = "This is the HST standard header packet
     containing observation parameters.  It is stored as a sequence of  965 two-byte integers.  For
    more detailed information, contact Space Telescope Science Institute."

    OBJECT                             = COLUMN
      NAME                             = PACKET_VALUES
      DATA_TYPE                        = MSB_INTEGER
      START_BYTE                       = 1
     BYTES                             = 2
    END_OBJECT                         = COLUMN
 END_OBJECT                            = HEADER_TABLE
END_OBJECT                             = UNCOMPRESSED_FILE
END
```

20.5 ZIPINFO.TXT Example

While the ZIPINFO.TXT file is not required, it is strongly recommended that this file be included as part of the process of documenting the contents of a zipfile. The following is an example ZIPINFO.TXT file and the type of information that should be included in the ZIPINFO.TXT file:

```
PDS_VERSION_ID                         = PDS3
RECORD_TYPE                            = STREAM

OBJECT                                 = TEXT
  PUBLICATION_DATE                     = 1999-07-26
  NOTE                                 = "This file provides an overview of the ZIP file format."

END_OBJECT                             = TEXT
END
```

Many of the files in this data set are compressed using Zip format. They are all indicated by the extension ".ZIP". ZIP is a utility that compresses files and also allows for multiple files to be stored in a single Zip archive. You will need the UNZIP utility to extract the files.

The SOFTWARE directory on this volume contains a complete description of the Zip file format and also the complete source code for the UNZIP utility. The file format and file decompression algorithms are described in the file SOFTWARE/APPNOTE.TXT.

It is far simpler to obtain a pre-built binary of the UNZIP application for your platform. Binaries for most platforms are available from the Info-ZIP web site, currently at:

http://www.cdrom.com/pub/infozip/

The same information can also be found a the PDS Central Node's web site, currently at:

http://pds.jpl.nasa.gov/

20.6 Additional Files

The PDS believes that Zip is a robust standard that will be in use for many years to come. Nevertheless, one cannot be certain that users in the distant future will have ready access to "Unzip" software for all future platforms. For this reason, any volume containing zipfiles is required to contain a complete description of the zipfile format, plus sample "Unzip" source code. This information must be found in a subdirectory of the SOFTWARE directory tree. This can be obtained from the Info-Zip web site, and the PDS Central Node will soon begin to maintain a sample SOFTWARE directory tree containing all the required information.

Appendix A

PDS Data Object Definitions

This section provides an alphabetical reference of PDS data object definitions, including a description, a list of required and optional keywords, a list of required and optional sub-objects (or child objects), and one or more examples.

NOTE: Any keywords in the Planetary Science Data Dictionary may also be included in the definition of a specific data object definition.

These definitions and examples are provided here for convenience. Additional examples of Data Object Definitions can be obtained by contacting your Data Engineer. As the definitions herein are subject to additions, modifications, and/or refinement, PDS has a web site where the current state of the Data Object Definitions can be ascertained:

http://pdsproto.jpl.nasa.gov/ddcolstdval/newdd/top.cfm

The examples provided in this Appendix have been based on both existing or planned PDS archive products, modified to reflect the most recent version of the PDS standards. They are not intended to represent existing data products and data object definitions designed under previous PDS standards.

The following PDS approved data object definitions are to be used for labeling primary and secondary data objects. For a more detailed discussion on primary and secondary data objects, see the Data Products chapter in this document.

There now exist four new Primitive Data Objects, ARRAY, BIT_ELEMENT (still under review), COLLECTION and ELEMENT. Although these objects are available, they should only be used after careful consideration of the current PDS Data Objects. Please see the PDS Objects chapter in this document for guidelines on the use of primitive objects.

TABLE OF CONTENTS

A.1 ALIAS

The ALIAS object provides a method for identifying alternate terms or names for approved data elements or objects within a data system. The ALIAS object is an optional sub-object of the COLUMN object.

Required Keywords

1. ALIAS_NAME
2. USAGE_NOTE

Optional Keywords

None

Required Objects

None

Optional Objects

None

Example

The following is an example of the usage of the ALIAS object as a subobject of COLUMN in a Magellan ARCDR label:

```
OBJECT                                    = COLUMN
  NAME                                    = ALT_FOOTPRINT_LONGITUDE
  START_BYTE                              = 1
  DATA_TYPE                               = REAL
  BYTES                                   = 10

  OBJECT                                  = ALIAS
    ALIAS_NAME                            = AR_LON
    USAGE_NOTE                            = "MAGELLAN MIT ARCDR SIS"
  END_OBJECT                              = ALIAS
END_OBJECT                                = COLUMN
```

A.2 ARRAY (Primitive Data Object)

The ARRAY object is provided to describe dimensioned arrays of homogeneous objects. Note that an ARRAY can contain only a single object, which can itself be another ARRAY or COLLECTION if required. A maximum of 6 axes is allowed in an ARRAY. The optional _AXIS_ elements can be used to describe the variation between successive objects in the ARRAY.

Values for AXIS_ITEMS and _AXIS_ elements for multidimensional arrays are supplied as sequences in which the right most item varies the fastest as the default.

The optional START_BYTE data element provides the starting location relative to an enclosing object. If a START_BYTE is not specified, a value of 1 is assumed.

Required Keywords

1. AXES
2. AXIS_ITEMS
3. NAME

Optional Keywords

1. AXIS_INTERVAL
2. AXIS_NAME
3. AXIS_UNIT
4. AXIS_START
5. AXIS_STOP
6. AXIS_ORDER_TYPE
7. CHECKSUM
8. DESCRIPTION
9. INTERCHANGE_FORMAT
10. START_BYTE

Required Objects

None

Optional Objects

1. ARRAY
2. BIT_ELEMENT
3. COLLECTION
4. ELEMENT

Example 1

The following is an example of a two dimensional Spectrum Array in a detached label.

```
PDS_VERSION_ID                  = PDS3
RECORD_TYPE                     = FIXED_LENGTH
RECORD_BYTES                    = 1600
FILE_RECORDS                    = 180

DATA_SET_ID                     = "IHW-C-SPEC-2-EDR-HALLEY-V1.0"
OBSERVATION_ID                  = "704283"
TARGET_NAME                     = "HALLEY"
INSTRUMENT_HOST_NAME            = "IHW SPECTROSCOPY AND SPECTROPHOTOMETRY NETWORK"
INSTRUMENT_NAME                 = "IHW SPECTROSCOPY AND SPECTROPHOTOMETRY"
PRODUCT_ID                      = "704283"
OBSERVATION_TIME                = 1986-05-09T04:10:20.640Z
START_TIME                      = 1986-05-09T04:07:50.640Z
STOP_TIME                       = UNK
PRODUCT_CREATION_TIME           = 1993-01-01T00:00:00.000Z
^ARRAY                          = "SPEC2702.DAT"
/* Description of Object in File */
OBJECT                          = ARRAY
NAME                            = "2D SPECTRUM"
INTERCHANGE_FORMAT              = BINARY
AXES                            = 2
AXIS_ITEMS                      = (180,800)
AXIS_NAME                       = ("RHO","APPROXIMATE WAVELENGTH")
AXIS_UNIT                       = (ARCSEC,ANGSTROMS)
AXIS_INTERVAL                   = (1.5,7.2164)
AXIS_START                      = (1.0,5034.9)

OBJECT                          = ELEMENT
DATA_TYPE                       = MSB_INTEGER
BYTES                           = 2
NAME                            = COUNT
DERIVED_MAXIMUM                 = 2.424980E+04
DERIVED_MINIMUM                 = 0.000000E+00
OFFSET                          = 0.000000E+00
SCALING_FACTOR                  = 1.000000E+00
NOTE                            = "Conversion factor 1.45 may be applied to data to estimate photons/sq m/sec/
                                    angstrom at 6800 angstroms."
END_OBJECT                      = ELEMENT
END_OBJECT                       = ARRAY
END
```

Example 2

The following is an example of ARRAY, COLLECTION and ELEMENT primitive objects all used together.

```
PDS_VERSION_ID                  = PDS3
RECORD_TYPE                     = FIXED_LENGTH
RECORD_BYTES                    = 122
FILE_RECORDS                    = 7387

^ARRAY                          = "MISCHA01.DAT"
```

```
DATA_SET_ID                          = "VEGA1-C-MISCHA-3-RDR-HALLEY-V1.0"
TARGET_NAME                          = HALLEY
SPACECRAFT_NAME                      = "VEGA 1"
INSTRUMENT_NAME                      = "MAGNETOMETER"
PRODUCT_ID                           = "XYZ"
START_TIME                           = "UNK"
STOP_TIME                            = "UNK"
SPACECRAFT_CLOCK_START_COUNT         = "UNK"
SPACECRAFT_CLOCK_STOP_COUNT          = "UNK"

NOTE                                 = "VEGA 1 MISCHA DATA"

OBJECT                               = ARRAY
NAME                                 = MISCHA_DATA_FILE
INTERCHANGE_FORMAT                   = BINARY
AXES                                 = 1
AXIS_ITEMS                           = 7387
DESCRIPTION                          = "This file contains an array of fixed length Mischa records."

OBJECT                               = COLLECTION
NAME                                 = MISCHA_RECORD
BYTES                                = 122
DESCRIPTION                          = "Each record in this file consists of a time tag followed by a  20-element array
                                         of magnetic field vectors."

OBJECT                               = ELEMENT
NAME                                 = START_TIME
BYTES                                = 2
DATA_TYPE                            = MSB_INTEGER
START_BYTE                           = 1
END_OBJECT                           = ELEMENT

OBJECT                               = ARRAY
NAME                                 = MAGNETIC_FIELD_ARRAY
AXES                                 = 2
AXIS_ITEMS                           = (3,20)
START_BYTE                           = 3
AXIS_NAME                            = ("XYZ_COMPONENT","TIME"  )
AXIS_UNIT                            = ("N/A"          ,"SECOND")
AXIS_INTERVAL                        = ("N/A"          , 0.2    )
DESCRIPTION                          = "Magnetic field vectors were recorded at the rate
of 10 per second. The START_TIME field gives the time at which the first vector in the record was recorded.  Successive vectors
were recorded  at 0.2 second intervals."

OBJECT                               = ELEMENT
NAME                                 = MAG_FIELD_COMPONENT_VALUE
BYTES                                = 2
DATA_TYPE                            = MSB_INTEGER
START_BYTE                           = 1
END_OBJECT                           = ELEMENT
END_OBJECT                           = ARRAY

END_OBJECT                           = COLLECTION

END_OBJECT                           = ARRAY
END
```

A.3 BIT COLUMN

The BIT_COLUMN object identifies a string of bits that do not fall on even byte boundaries and therefore cannot be described as a distinct COLUMN. BIT_COLUMNS defined within columns are analogous to columns defined within rows.

Note: (1) The Planetary Data System recommends that all fields (within new objects) should be defined on byte boundaries. This precludes having multiple values strung together in bit strings, as occurs in the BIT_COLUMN object.

(2) BIT_COLUMN is intended for use in describing existing binary data strings, but is not recommended for use in defining new data objects because it will not be recognized by most general purpose software.

(3) A BIT_COLUMN must not contain embedded objects.

BIT_COLUMNS of the same format and size may be specified as a single BIT_COLUMN by using the ITEMS, ITEM_BITS, and ITEM_OFFSET elements. The ITEMS data element is used to indicate the number of occurrences of a bit string.

Required Keywords

1. NAME
2. BIT_DATA_TYPE
3. START_BIT
4. BITS (required for BIT_COLUMNs without items)
5. DESCRIPTION

Optional Keywords

1. BIT_MASK
2. BITS (optional for BIT_COLUMNs with items)
3. FORMAT
4. INVALID_CONSTANT
5. ITEMS
6. ITEM_BITS
7. ITEM_OFFSET
8. MINIMUM
9. MAXIMUM
10. MISSING_CONSTANT
11. OFFSET
12. SCALING_FACTOR
13. UNIT

Required Objects

None

Optional Objects

None

Example

The example below was extracted from a larger example which can be found within the CONTAINER object. The BIT_COLUMN object can be a sub-object of the TABLE or CONTAINER object.

```
OBJECT                              =COLUMN
NAME                                =PACKET_ID
DATA_TYPE                           =LSB_BIT_STRING
START_BYTE                          =1
BYTES                               =2
VALID_MINIMUM                       =0
VALID_MAXIMUM                       =7
DESCRIPTION                         = "Packet_id constitutes one of three parts  in the  primary source information
header applied by the Payload Data System (PDS) to the MOLA telemetry packet at the time of creation of the packet prior to
transfer frame creation.  "

OBJECT                              =BIT_COLUMN
NAME                                =VERSION_NUMBER
BIT_DATA_TYPE                       =MSB_UNSIGNED_INTEGER
START_BIT                           =1
BITS                                =3
MINIMUM                             =0
MAXIMUM                             =7
DESCRIPTION                         = "These bits identify Version 1 as the Source Packet structure.  These bits shall
be set to '000'."
END_OBJECT                          =BIT_COLUMN

OBJECT                              =BIT_COLUMN
NAME                                =SPARE
BIT_DATA_TYPE                       =MSB_UNSIGNED_INTEGER
START_BIT                           =4
BITS                                =1
MINIMUM                             =0
MAXIMUM                             =0
DESCRIPTION                         ="Reserved spare.  This bit shall be set to '0'"
END_OBJECT                          =BIT_COLUMN

OBJECT                              =BIT_COLUMN
NAME                                =FLAG
BIT_DATA_TYPE                       =BOOLEAN
START_BIT                           =5
BITS                                =1
MINIMUM                             =0
MAXIMUM                             =0
DESCRIPTION                         ="This flag signals the presence or absence of a Secondary Header data structure
within the Source Packet.   This bit shall be set to '0' since no Secondary Header formatting standards currently exist for Mars
Observer."
END_OBJECT                          =BIT_COLUMN

OBJECT                              =BIT_COLUMN
NAME                                =ERROR_STATUS
```

```
BIT_DATA_TYPE                         =MSB_UNSIGNED_INTEGER
START_BIT                             =6
BITS                                  =3
MINIMUM                               =0
MAXIMUM                               =7
DESCRIPTION                           ="This field identifies in part the individual application process within the
spacecraft that created the Source Packet data."
END_OBJECT                            = BIT_COLUMN

OBJECT                                = BIT_COLUMN
NAME                                  = INSTRUMENT_ID
BIT_DATA_TYPE                         = MSB_UNSIGNED_INTEGER
START_BIT                             = 9
BITS                                  = 8
MINIMUM                               = "N/A"
MAXIMUM                               = "N/A"
DESCRIPTION                           =  "This field identifies in part the individual application process within the
spacecraft that created the Source Packet data.  00100011 is the bit pattern for MOLA."
END_OBJECT                            = BIT_COLUMN
END_OBJECT                            = COLUMN
```

A.4 BIT ELEMENT (Primitive Data Object)

Under review.

A.5 CATALOG

The CATALOG object is used within a VOLUME object to reference completed PDS high level catalog templates. These templates provide additional information related to the data sets on the volume. Please refer to the File Specification and Naming chapter in this document for more information.

Required Keywords

None

Optional Keywords

1. DATA_SET_ID
2. LOGICAL_VOLUME_PATHNAME
3. LOGICAL_VOLUMES

Required Objects

1. DATA_SET
2. INSTRUMENT
3. INSTRUMENT_HOST
4. MISSION

Optional Objects

1. DATA_SET_COLLECTION
2. PERSONNEL
3. REFERENCE
4. TARGET

Example

The example under the VOLUME object provides an example of a CATALOG object where all the Catalog Templates are included in a single file, CATALOG.CAT.

The example below is a VOLDESC.CAT file that demonstrates multiple data sets per volume. In this example, the Catalog Templates are in separate files and are referenced by the use of pointers. However, the catalog templates may also be included in-line - but this is not the recommended approach (see Section 19.3, PDS Preferred Method for Supplying Catalog Objects).

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                        = PDS3
LABEL_REVISION_NOTE                   ="RSimpson, 1998-07-01"
RECORD_TYPE                           = STREAM

OBJECT                                = VOLUME
```

```
VOLUME_SERIES_NAME              = "VOYAGERS TO THE OUTER PLANETS"
VOLUME_SET_NAME                 = "VOYAGER NEPTUNE PLANETARY PLASMA INTERACTIONS DATA"
VOLUME_SET_ID                   = USA_NASA_PDS_VG_1001
VOLUMES                         = 1
VOLUME_NAME                     = "VOYAGER NEPTUNE PLANETARY PLASMA INTERACTIONS DATA"
VOLUME_ID                       = VG_1001
VOLUME_VERSION_ID               = "VERSION 1"
VOLUME_FORMAT                   = "ISO-9660"
MEDIUM_TYPE                     = "CD-ROM"
PUBLICATION_DATE                = 1992-11-13
DESCRIPTION                     = "This volume contains a collection of non-imaging Planetary Plasma datasets
from the Voyager 2 spacecraft encounter with Neptune.  Included are datasets from the Cosmic Ray System (CRS), Plasma System
(PLS), Plasma Wave System (PWS), Planetary Radio Astronomy (PRA), Magnetometer (MAG), and Low Energy Charged Particle
(LECP) instruments, as well as spacecraft position vectors (POS) in several coordinate systems.  The volume also contains
documentation and index files to support access and use of the data."

DATA_SET_ID                     = { "VG2-N-CRS-3-RDR-D1-6SEC-V1.0",
                                    "VG2-N-CRS-4-SUMM-D1-96SEC-V1.0",
                                    "VG2-N-CRS-4-SUMM-D2-96SEC-V1.0",
                                    "VG2-N-LECP-4-SUMM-SCAN-24SEC-V1.0",
                                    "VG2-N-LECP-4-RDR-STEP-12.8MIN-V1.0",
                                    "VG2-N-MAG-4-RDR-HG-COORDS-1.92SEC-V1.0",
                                    "VG2-N-MAG-4-SUMM-HG-COORDS-48SEC-V1.0",
                                    "VG2-N-MAG-4-RDR-HG-COORDS-9.6SEC-V1.0",
                                    "VG2-N-MAG-4-SUMM-NLSCOORDS-12SEC-V1.0",
                                    "VG2-N-PLS-5-RDR-2PROMAGSPH-48SEC-V1.0",
                                    "VG2-N-PLS-5-RDR-ELEMAGSPHERE-96SEC-V1.0",
                                    "VG2-N-PLS-5-RDR-IONMAGSPHERE-48SEC-V1.0",
                                    "VG2-N-PLS-5-RDR-IONLMODE-48SEC-V1.0",
                                    "VG2-N-PLS-5-RDR-IONMMODE-12MIN-V1.0",
                                    "VG2-N-PLS-5-RDR-ION-INBNDWIND-48SEC-V1.0",
                                    "VG2-N-POS-5-RDR-HGHGCOORDS-48SEC-V1.0",
                                    "VG2-N-POS-5-SUMM-NLSCOORDS-12-48SEC-V1.0",
                                    "VG2-N-PRA-4-SUMM-BROWSE-SEC-V1.0",
                                    "VG2-N-PRA-2-RDR-HIGHRATE-60MS-V1.0",
                                    "VG2-N-PWS-2-RDR-SA-4SEC-V1.0",
                                    "VG2-N-PWS-4-SUMM-SA-48SEC-V1.0",
                                    "VG2-N-PWS-1-EDR-WFRM-60MS-V1.0"}

OBJECT                          = DATA_PRODUCER
INSTITUTION_NAME                = "UNIVERSITY OF CALIFORNIA, LOS ANGELES"
FACILITY_NAME                   = "PDS PLANETARY PLASMA INTERACTIONS NODE"
FULL_NAME                       = "DR. RAYMOND WALKER"
DISCIPLINE_NAME                 = "PLASMA INTERACTIONS"
ADDRESS_TEXT                    = "UCLA
                                  IGPP
                                  LOS ANGELES, CA 90024  USA"
END_OBJECT                      = DATA_PRODUCER

OBJECT                          = DATA_SUPPLIER
INSTITUTION_NAME                = "NATIONAL SPACE SCIENCE DATA CENTER"
FACILITY_NAME                   = "NATIONAL SPACE SCIENCE DATA CENTER"
FULL_NAME                       = "NATIONAL SPACE SCIENCE DATA CENTER"
DISCIPLINE_NAME                 = "NATIONAL SPACE SCIENCE DATA CENTER"
ADDRESS_TEXT                    = "Code 633  \n
                                     Goddard Space Flight Center  \n
                                     Greenbelt, Maryland, 20771, USA"
TELEPHONE_NUMBER                = "3012866695"
ELECTRONIC_MAIL_TYPE            = "NSI/DECNET"
```

```
ELECTRONIC_MAIL_ID              = "NSSDCA::REQUEST"
END_OBJECT                      = DATA_SUPPLIER

OBJECT                          = CATALOG
^MISSION_CATALOG                = "MISSION.CAT"
^INSTRUMENT_HOST_CATALOG        = "INSTHOST.CAT"
^INSTRUMENT_CATALOG             = {"CRS_INST.CAT",
                                   "LECPINST.CAT",
                                   "MAG_INST.CAT",
                                   "PLS_INST.CAT",
                                   "PRA_INST.CAT",
                                   "PWS_INST.CAT"}
^DATA_SET_CATALOG               = {"CRS_DS.CAT",
                                   "LECP_DS.CAT",
                                   "MAG_DS.CAT",
                                   "PLS_DS.CAT",
                                   "POS_DS.CAT",
                                   "PRA_DS.CAT",
                                   "PWS_DS.CAT"}
^TARGET_CATALOG                 = TGT.CAT
^PERSONNEL_CATALOG              = PERS.CAT
^REFERENCE_CATALOG              = REFS.CAT
END_OBJECT                      = CATALOG

END_OBJECT                      = VOLUME
END
```

A.6 COLLECTION (Primitive Data Object)

The COLLECTION object allows the ordered grouping of heterogeneous objects into a named collection. The COLLECTION object may contain a mixture of different object types including other COLLECTIONS. The optional START_BYTE data element provides the starting location relative to an enclosing object. If a START_BYTE is not specified, a value of 1 is assumed.

Required Keywords

1. BYTES
2. NAME

Optional Keywords

1. DESCRIPTION
2. CHECKSUM
3. INTERCHANGE_FORMAT
4. START_BYTE

Required Objects

None

Optional Objects

1. ELEMENT
2. BIT_ELEMENT
3. ARRAY
4. COLLECTION

Example

Please refer to the example in the ARRAY Primitive object for an example of an implementation of the COLLECTION object.

A.7 COLUMN

The COLUMN object identifies a single column in a data object.

Note:(1) Current PDS-described data objects that include COLUMN objects are the TABLE, CONTAINER, SPECTRUM and SERIES objects.
(2) COLUMNs must not contain embedded COLUMN objects.
(3) COLUMNs of the same format and size may be specified as a single COLUMN by using the ITEMS, ITEM_BYTES, and ITEM_OFFSET elements. The ITEMS data element indicates the number of occurrences of the field.
(4) BYTES and ITEM_BYTES counts do not include leading or trailing delimiters or line terminators.
(5) For a COLUMN with items, the value of BYTES should represent the size of the column including delimiters between the items. See examples 1 and 2 below.

Required Keywords

1. NAME
2. DATA_TYPE
3. START_BYTE
4. BYTES (required for COLUMNs without items)

Optional Keywords

1. BIT_MASK
2. BYTES (optional for COLUMNs with items)
3. DERIVED_MAXIMUM
4. DERIVED_MINIMUM
5. DESCRIPTION
6 . FORMAT
7. INVALID_CONSTANT
8. ITEM_BYTES
9. ITEM_OFFSET
10. ITEMS
11. MAXIMUM
12. MAXIMUM_SAMPLING_PARAMETER
13. MINIMUM
14. MINIMUM_SAMPLING_PARAMETER
15. MISSING_CONSTANT
16. OFFSET
17. SAMPLING_PARAMETER_INTERVAL
18. SAMPLING_PARAMETER_NAME
19. SAMPLING_PARAMETER_UNIT
20. SCALING_FACTOR
21. UNIT
22. VALID_MAXIMUM
23. VALID_MINIMUM

Required Objects

None

Optional Objects

1. BIT_COLUMN
2. ALIAS

Example 1

The example below shows the use of a COLUMN with items. In this example, the data described is a column with three ASCII_INTEGER items: xx,yy, zz

The ITEM_OFFSET is the number of bytes from the beginning of one item to the beginning of the next.

Note that the value of BYTES includes the comma delimiters between items.

```
OBJECT                          = COLUMN
NAME                            = COLUMNXYZ
DATA_TYPE                       = ASCII_INTEGER
START_BYTE                      = 1
BYTES                           = 8  /*includes delimiters*/
ITEMS                           = 3
ITEM_BYTES                      = 2
ITEM_OFFSET                     = 3
END_OBJECT                      =  COLUMN
```

Example 2

The example below again shows the use of a COLUMN with items. In this example, the data described is a column with three CHARACTER items: "xx", "yy", "zz"

```
OBJECT                          =  COLUMN
NAME                            =  COLUMNXYZ
DATA_TYPE                       =  CHARACTER
START_BYTE                      =  2                   /* value does not include leading quote */
BYTES                           =  12                   /* value does not include leading and trailing
                                                         quotes */
ITEMS                           =  3
ITEM_BYTES                      =  2                   /* value does not include leading and trailing
                                                       quotes */
ITEM_OFFSET                     =  5                   /* value does not include leading quote */
END_OBJECT                      =  COLUMN
```

Example 3

The example below was extracted from a larger example which can be found under the CONTAINER object. The COLUMN object is a sub-object of the TABLE, SERIES, SPECTRUM, and CONTAINER objects.

```
OBJECT                          = COLUMN
NAME                            = PACKET_ID
DATA_TYPE                       = LSB_BIT_STRING
START_BYTE                      = 1
BYTES                           = 2
VALID_MINIMUM                   = 0
VALID_MAXIMUM                   = 7
DESCRIPTION                     = "Packet_id constitutes one of three parts in the  primary source information
header applied by the Payload Data System (PDS) to the MOLA telemetry packet at the time of creation of the packet prior to
transfer frame creation.  "

OBJECT                          = BIT_COLUMN
NAME                            = VERSION_NUMBER
BIT_DATA_TYPE                   = MSB_UNSIGNED_INTEGER
START_BIT                       = 1
BITS                            = 3
MINIMUM                         = 0
MAXIMUM                         = 7
DESCRIPTION                     =  "These bits identify Version 1 as the Source Packet structure.  These bits
shall be set to '000'."
END_OBJECT                      =   BIT_COLUMN

OBJECT                          = BIT_COLUMN
NAME                            = SPARE
BIT_DATA_TYPE                   = MSB_UNSIGNED_INTEGER
START_BIT                       = 4
BITS                            = 1
MINIMUM                         = 0
MAXIMUM                         = 0
DESCRIPTION                     = "Reserved spare.  This bit shall be set to '0'"
END_OBJECT                      =  BIT_COLUMN

OBJECT                          = BIT_COLUMN
NAME                            = FLAG
BIT_DATA_TYPE                   = BOOLEAN
START_BIT                       = 5
BITS                            = 1
MINIMUM                         = 0
MAXIMUM                         = 0
DESCRIPTION                     =  "This flag signals the presence or absence of a Secondary Header data structure
within the Source Packet. This bit shall be set to '0' since no Secondary Header formatting standards currently exist for Mars
Observer."
END_OBJECT                      = BIT_COLUMN

OBJECT                          = BIT_COLUMN
NAME                            = ERROR_STATUS
BIT_DATA_TYPE                   = MSB_UNSIGNED_INTEGER
START_BIT                       = 6
BITS                            = 3
MINIMUM                         = 0
MAXIMUM                         = 7
DESCRIPTION                     = "This field identifies in part the individual application process within the
```

```
spacecraft that created the Source Packet data."
END_OBJECT                              = BIT_COLUMN

OBJECT                                  = BIT_COLUMN
NAME                                    = INSTRUMENT_ID
BIT_DATA_TYPE                           = MSB_UNSIGNED_INTEGER
START_BIT                               = 9
BITS                                    = 8
MINIMUM                                 = "N/A"
MAXIMUM                                 = "N/A"
DESCRIPTION                             = "This field identifies in part the individual application process within the
spacecraft that creeated the Source Packet data.  00100011 is the bit pattern for MOLA."
END_OBJECT                              = BIT_COLUMN
END_OBJECT                              = COLUMN

OBJECT                                  = COLUMN
NAME                                    = CH_4_2ND_HALF_FRAME_BKGRND_CN
DATA_TYPE                               = UNSIGNED_INTEGER
START_BYTE                              = 134
BYTES                                   = 1
MINIMUM                                 = 0
MAXIMUM                                 = 255
DESCRIPTION                             =  "The background energy or noise count levels in channels 1, 2, 3, and 4
respectively by half-frame.  Pseudo log value of NOISE(1, 2, 3, 4) at the end of a half-frame of current frame, 5.3 bit format.  Plog
base 2 of background count sum..."
END_OBJECT                              = COLUMN
```

A.8 CONTAINER

The CONTAINER object is used to group a set of sub-objects (such as COLUMNS) that repeat within a data object (such as a TABLE). Use of the CONTAINER object allows repeating groups to be defined within a data structure.

Required Keywords

1. NAME
2. START_BYTE
3. BYTES
4. REPETITIONS
5. DESCRIPTION

Optional Keywords

None

Required Objects

None

Optional Objects

1. COLUMN
2. CONTAINER

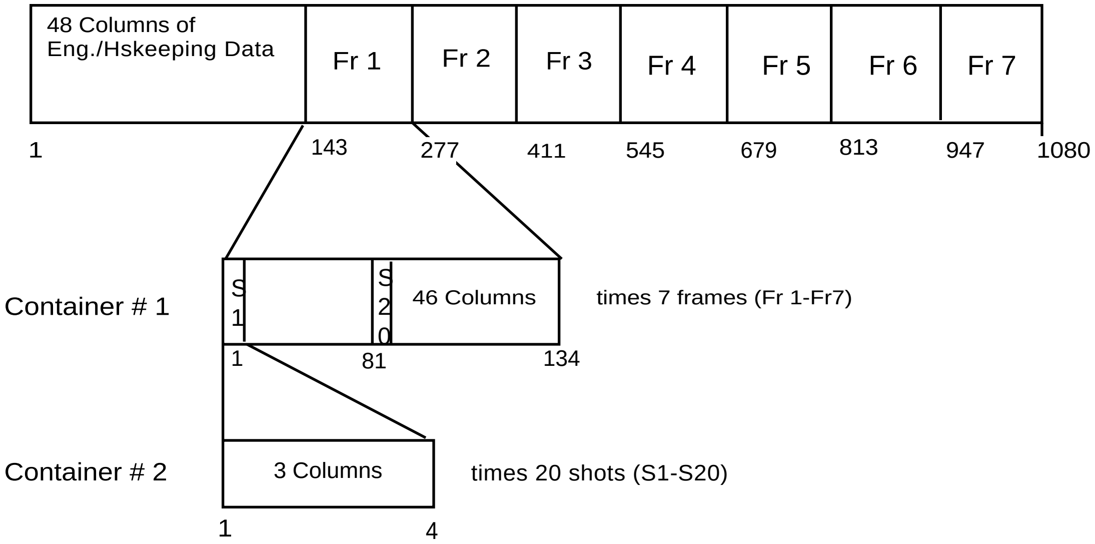
Example

The following diagram shows a data product layout in which the CONTAINER object is used. The diagram depicts the modelled data product as a TABLE with one row (or one record of data). Each record within the diagram begins with 48 columns (143 bytes) of engineering data. The data product acquires science data from seven different frames. Since the data from each frame are formatted identically, one CONTAINER description can suffice for all seven frames.

In this example there are two CONTAINER objects. The first CONTAINER object describes the repeating frame information. Within this CONTAINER there is a second CONTAINER object in which a 4-byte set of three COLUMN objects repeats 20 times. The use of the second CONTAINER object permits the data supplier to describe the three COLUMNS (4 bytes) once, instead of specifying sixty column definitions.

In the first CONTAINER, the keyword REPETITIONS is equal to 7. In the second CONTAINER, REPETITIONS equals 20. Both CONTAINER objects contain a collection of COLUMN objects. In most cases it is preferable to save space in the product label by placing COLUMN objects in a separate file and pointing to that file from within the CONTAINER object.

.



This attached label example describes the above TABLE structure using CONTAINER objects.

```
CCSD3ZF0000100000001NJPL3KS0PDSXAAAAAAAA
PDS_VERSION_ID                      = PDS3
RECORD_TYPE                         = FIXED_LENGTH
FILE_RECORDS                        = 467
RECORD_BYTES                        = 1080
LABEL_RECORDS                       = 4
FILE_NAME                           =  "AEDR.001"

^MOLA_SCIENCE_MODE_TABLE            = 5
DATA_SET_ID                         =  "MO-M-MOLA-1-AEDR-L0-V1.0"
PRODUCT_ID                          =  "MOLA-AEDR.-10010-0001"
SPACECRAFT_NAME                     =   MARS_OBSERVER
INSTRUMENT_ID                       =   MOLA
INSTRUMENT_NAME                     =   MARS_OBSERVER_LASER_ALTIMETER
TARGET_NAME                         =   MARS
SOFTWARE_NAME                       =   "Browser 17.1"
UPLOAD_ID                           =   "5.3"
PRODUCT_RELEASE_DATE                =   1994-12-29T02:10:09.321
START_TIME                          =   1994-09-29T04:12:43.983
STOP_TIME                           =   1994-09-29T06:09:54.221
SPACECRAFT_CLOCK_START_COUNT        =   "12345"
SPACECRAFT_CLOCK_STOP_COUNT         =   "12447"
PRODUCT_CREATION_TIME               =   1995-01-29T07:30:333
MISSION_PHASE_NAME                  =   MAPPING
ORBIT_NUMBER                        =   0001
PRODUCER_ID                         =   MO_MOLA_TEAM
PRODUCER_FULL_NAME                  =   "DAVID E. SMITH"
PRODUCER_INSTITUTION_NAME           =   "GODDARD SPACE FLIGHT CENTER"
DESCRIPTION                         =   "This data product contains the aggregation of MOLA telemetry packets by
Orbit.  All Experiment Data Record Packets retrieved from the PDB are collected in this data product.  The AEDR data product is
put together with the Project-provided software tool Browser."
```

```
OBJECT                          =  MOLA_SCIENCE_MODE_TABLE
INTERCHANGE_FORMAT              =  BINARY
ROWS                            =  463
COLUMNS                         =  97
ROW_BYTES                       =  1080
^STRUCTURE                      =  "MOLASCI.FMT"
DESCRIPTION                     =  "This table is one of two that describe the arrangement of information on the
Mars Observer Laser Altimeter (MOLA) Aggregated Engineering Data Record (AEDR).  ..."

END_OBJECT                      =     MOLA_SCIENCE_MODE_TABLE
...

END
CCSD$$MARK$$AAAAAAAANJPL3IF0NNNN00000001

Contents of the MOLASCI.FMT file
--------------------------------------------------------------------------------------------------------------

OBJECT                          =  COLUMN
NAME                            =  PACKET_ID
DATA_TYPE                       =  LSB_BIT_STRING
START_BYTE                      =  1
BYTES                           =  2
VALID_MINIMUM                   =  0
VALID_MAXIMUM                   =  7
DESCRIPTION                     =  "Packet_id constitutes one of three parts in the  primary source information
header applied by the Payload Data System (PDS) to the MOLA telemetry packet at the time of creation of the packet prior to
transfer frame creation.  "

OBJECT                          =  BIT_COLUMN
NAME                            =  VERSION_NUMBER
BIT_DATA_TYPE                   =  UNSIGNED_INTEGER
START_BIT                       =  1
BITS                            =  3
MINIMUM                         =  0
MAXIMUM                         =  7
DESCRIPTION                     =   "These bits identify Version 1 as the Source Packet structure.  These bits shall
be set to '000'."
END_OBJECT                      =  BIT_COLUMN

OBJECT                          =  BIT_COLUMN
NAME                            =  SPARE
BIT_DATA_TYPE                   =  UNSIGNED_INTEGER
START_BIT                       =  4
BITS                            =  1
MINIMUM                         =  0
MAXIMUM                         =  0
DESCRIPTION                     =  "Reserved spare.  This bit shall be set to '0'"
END_OBJECT                      =  BIT_COLUMN

OBJECT                          =  BIT_COLUMN
NAME                            =  SECONDARY_HEADER_FLAG
BIT_DATA_TYPE                   =  BOOLEAN
START_BIT                       =  5
BITS                            =  1
MINIMUM                         =  0
MAXIMUM                         =  0
DESCRIPTION                     =  "This flag signals the presence or absence of a Secondary Header data
structure within the Source Packet. This bit shall be set to '0' since no Secondary Header formatting standards currently exist for
```

```
Mars Observer."
END_OBJECT                     =  BIT_COLUMN

OBJECT                         =  BIT_COLUMN
NAME                           =  ERROR_STATUS
BIT_DATA_TYPE                  =  UNSIGNED_INTEGER
START_BIT                      =   6
BITS                           =  3
MINIMUM                        =  0
MAXIMUM                        =  7
DESCRIPTION                    =   "This field identifies in part the individual application process within the
spacecraft that created the Source Packet data."
END_OBJECT                     =  BIT_COLUMN

OBJECT                         =  BIT_COLUMN
NAME                           =  INSTRUMENT_ID
BIT_DATA_TYPE                  =  UNSIGNED_INTEGER
START_BIT                      =  9
BITS                           =  8
MINIMUM                        =  2#0100011#
MAXIMUM                        =  2#0100011#
DESCRIPTION                    =    "This field identifies in part the individual application process within the
spacecraft that created the Source Packet data.  00100011 is the bit pattern for MOLA."
END_OBJECT                     =  BIT_COLUMN
END_OBJECT                     =  COLUMN

...

OBJECT                         = COLUMN
NAME                           =  COMMAND_ECHO
DATA_TYPE                      =  INTEGER
START_BYTE                     = 125
BYTES                          = 16
ITEMS                          =  8
ITEM_BYTES                     =  2
MINIMUM                        =  0
MAXIMUM                        =  65535
DESCRIPTION                    =  "First 8 command words received during current packet, only complete
commands are stored, MOLA specific commands only. The software attempts to echo all valid commands.  If the command will fit
in the room remaining in the..."
END_OBJECT                     =  COLUMN

OBJECT                         =  COLUMN
NAME                           =  PACKET_VALIDITY_CHECKSUM
DATA TYPE                      =  INTEGER
START_BYTE                     =  141
BYTES                          =  2
MINIMUM                        =  0
MAXIMUM                        =  65535
DESCRIPTION                    =   "Simple 16 bit addition of entire packet contents upon completion.  This
location is zeroed for addition. This word is zeroed, then words 0-539 are added without carry to a variable that is initially zero. The
resulting lower 16 bits ar..."
END_OBJECT                     =  COLUMN

OBJECT                         =  CONTAINER
NAME                           =  FRAME_STRUCTURE
^STRUCTURE                     =   "MOLASCFR.FMT"  /*points to the columns */
 /*that make up the frame descriptors */
START_BYTE                     =  143
```

```
BYTES                          =  134
REPETITIONS                    =  7
DESCRIPTION                    =  "The frame_structure container represents the format of seven repeating
groups of attributes in this data product.  The data product reflects science data acquisition from seven different frames.  Since the
data from each frame are ..."
END_OBJECT                     =  CONTAINER

CONTENTS OF THE MOLASCFR.FMT FILE
------------------------------------------------------------------------------------------------------------

OBJECT                         =  CONTAINER
NAME                           =  COUNTS
START_BYTE                     =  1
BYTES                          =  4
REPETITIONS                    =  20
^STRUCTURE                     =  "MOLASCCT.FMT"
DESCRIPTION                    =  "This container has three sub-elements (range to surface counts, 1st channel
received pulse energy, and 2nd channel received pulse energy).  The three sub-elements repeat for each of 20 shots."
END_OBJECT                     =  CONTAINER

OBJECT                         =  COLUMN
NAME                           =  SHOT_2_LASER_TRANSMITTER_POWR
DATA_TYPE                      =  UNSIGNED_INTEGER
START_BYTE                     =  81
BYTES                          =  1
MINIMUM                        =  0
MAXIMUM                        =  65535
DESCRIPTION                    =  "..."
END_OBJECT                     =  COLUMN

OBJECT                         =  COLUMN
NAME                           =  SHOT_1_LASER_TRANSMITTER_POWR
DATA_TYPE                      =  UNSIGNED_INTEGER
START_BYTE                     =  82
BYTES                          =  1
MINIMUM                        =  0
MAXIMUM                        =  65535
DESCRIPTION                    =   "..."
END_OBJECT                     =  COLUMN

OBJECT                         =  COLUMN
NAME                           =  SHOT_4_LASER_TRANSMITTER_POWR
DATA_TYPE                      =  UNSIGNED_INTEGER
START_BYTE                     =  83
BYTES                          =  1
MINIMUM                        =  0
MAXIMUM                        =  65535
DESCRIPTION                    =  "..."
END_OBJECT                     =  COLUMN

...

OBJECT                         =  COLUMN
NAME                           =  CH_3_2ND_HALF_FRAME_BKGRND_CN
DATA_TYPE                      =  UNSIGNED_INTEGER
START_BYTE                     =  133
BYTES                          =  1
MINIMUM                        =  0
MAXIMUM                        =  255
```

```
DESCRIPTION                          =   "The background energy or noise count levels in channels 1, 2, 3, and 4
respectively by half-frame.  Pseudo log value of NOISE(1, 2, 3, 4) at the end of a half-frame of current frame, 5.3 bit format.  Plog
base 2 of background count sum..."
END_OBJECT                           =  COLUMN

OBJECT                               =  COLUMN
NAME                                 =  CH_4_2ND_HALF_FRAME_BKGRND_CN
DATA_TYPE                            =  UNSIGNED_INTEGER
START_BYTE                           =  134
BYTES                                =  1
MINIMUM                              =  0
MAXIMUM                              =  255
DESCRIPTION                          =   "The background energy or noise count levels in channels 1, 2, 3, and 4
respectively by half-frame.  Pseudo log value of NOISE(1, 2, 3, 4) at the end of a half-frame of current frame, 5.3 bit format.  Plog
base 2 of background count sum..."
END_OBJECT                           =  COLUMN

CONTENTS OF THE MOLASCCT.FMT FILE
------------------------------------------------------------------------------------------------------

OBJECT                               =  COLUMN
NAME                                 =  RANGE_TO_SURFACE_TIU_CNTS
DATA_TYPE                            =  MSB_INTEGER
START_BYTE                           =  1
BYTES                                =  2
DESCRIPTION                          =    "The possible 20 valid frame laser shots surface ranging measurements in
Timing Interval Unit (TIU) counts.  The least significant 16 bits of TIU (SLTIU), stored for every shot. B[0] = Bits 15-8 of TIU
reading; B[1] = Bits 7-0 of ..."
END_OBJECT                           =  COLUMN

OBJECT                               =  COLUMN
NAME                                 =  FIRST_CH_RCVD_PULSE_ENRGY
DATA_TYPE                            =  UNSIGNED_INTEGER
START_BYTE                           =  3
BYTES                                =  1
DESCRIPTION                          =   "The level of return, reflected energy as received by the first channel and
matched filter to trigger.  This is a set of  values for all possible 20 shots within the frame.  Lowest numbered non-zero energy
reading for each shot."
END_OBJECT                           =  COLUMN

OBJECT                               =  COLUMN
NAME                                 =  SECOND_CH_RCVD_PULSE_ENRGY
DATA_TYPE                            =  UNSIGNED_INTEGER
START_BYTE                           =  4
BYTES                                =  1
DESCRIPTION                          =   "The level of return, reflected energy as received by the second channel and
matched filter to trigger.  This is a set of values for all possible 20 shots within the frame.  2nd lowest numbered non-zero energy
reading for each shot..."
END_OBJECT                           =  COLUMN
```

A.9 DATA PRODUCER

The DATA_PRODUCER object is used within a PDS object, such as VOLUME. The DATA_PRODUCER, as opposed to the DATA_SUPPLIER, is an individual or organization responsible for collecting, assembling, and/or engineering the raw data into one or more data sets.

Required Keywords

1. INSTITUTION_NAME
2. FACILITY_NAME
3. FULL_NAME
4. ADDRESS_TEXT

Optional Keywords

1. DISCIPLINE_NAME
2. NODE_NAME
3. TELEPHONE_NUMBER
4. ELECTRONIC_MAIL_TYPE
5. ELECTRONIC_MAIL_ID

Required Objects

None

Optional Objects

None

Example

The example below was extracted from a larger example which can be found within the VOLUME object. The DATA_PRODUCER object is a required object of the VOLUME.

```
OBJECT                          = DATA_PRODUCER
INSTITUTION_NAME                = "U.S.G.S. FLAGSTAFF"
FACILITY_NAME                   = "BRANCH OF ASTROGEOLOGY"
FULL_NAME                       = "Eric M. Eliason"
DISCIPLINE_NAME                 = "IMAGE PROCESSING"
ADDRESS_TEXT                    = " Branch of Astrogeology
                                  United States Geological Survey
                                  2255 North Gemini Drive
                                  Flagstaff, Arizona. 86001 USA"
END_OBJECT                      = DATA_PRODUCER
```

A.10 DATA SUPPLIER

The DATA_SUPPLIER object is used within a PDS object, such as VOLUME. The DATA_SUPPLIER, as opposed to the DATA_PRODUCER, is an individual or organization responsible for distributing the data sets and associated data to the science community.

Required Keywords

1. INSTITUTION_NAME
2. FACILITY_NAME
3. FULL_NAME
4. ADDRESS_TEXT
5. TELEPHONE_NUMBER
6. ELECTRONIC_MAIL_TYPE
7. ELECTRONIC_MAIL_ID

Optional Keywords

1. DISCIPLINE_NAME
2. NODE_NAME

Required Objects

None

Optional Objects

None

Example

The example below was extracted from a larger example which can be found within the VOLUME object. The DATA_SUPPLIER object is an optional object of the VOLUME.

```
OBJECT                          = DATA_SUPPLIER
INSTITUTION_NAME                = "NATIONAL SPACE SCIENCE DATA CENTER"
FACILITY_NAME                   = "NATIONAL SPACE SCIENCE DATA CENTER"
FULL_NAME                       = "NATIONAL SPACE SCIENCE DATA CENTER"
DISCIPLINE_NAME                 = "NATIONAL SPACE SCIENCE DATA CENTER"
ADDRESS_TEXT                    = "Code 633
                                  Goddard Space Flight Center
                                  Greenbelt, Maryland, 20771, USA"
 TELEPHONE_NUMBER               = "3012866695"
ELECTRONIC_MAIL_TYPE            = "NSI/DECNET"
ELECTRONIC_MAIL_ID              = "NSSDCA::REQUEST"
END_OBJECT                      = DATA_SUPPLIER
```

A.11 DIRECTORY

The DIRECTORY object is used to define a hierarchical file organization on a linear (sequential) media, such as tape. The DIRECTORY object identifies all directories and subdirectories below the root level, and is a required sub-object of the VOLUME object for tape media.

Note: The root directory on a volume does not need to be explicitly defined with the DIRECTORY object.

Subdirectories are identified by embedding DIRECTORY objects. Files within the directories and subdirectories are sequentially identified by using FILE objects with a sequence_number value corresponding to their position on the media. A sequence_number value will be unique for each file on the media. This format is strongly recommended when transferring or archiving volumes of data on media which do not support hierarchical directory structures (i.e., submitting a tape volume of data for pre-mastering or preparing an archive tape).

Although the DIRECTORY object is optional in the VOLUME object, it is a required object for tape media.

Required Keywords

1. NAME

Optional Keywords

1. RECORD_TYPE
2. SEQUENCE_NUMBER

Required Objects

1. FILE

Optional Objects

1. DIRECTORY

Example

The example below was extracted from a larger example which can be found within the VOLUME object.

```
OBJECT                          = DIRECTORY
NAME                            = INDEX

OBJECT                          = FILE
FILE_NAME                       = "INDXINFO.TXT"
RECORD_TYPE                     = STREAM
SEQUENCE_NUMBER                 = 5
END_OBJECT                      = FILE

OBJECT                          = FILE
FILE_NAME                       = "INDEX.LBL"
RECORD_TYPE                     = STREAM
SEQUENCE_NUMBER                 = 6
END_OBJECT                      = FILE

OBJECT                          = FILE
FILE_NAME                       = "INDEX.TAB"
RECORD_TYPE                     = FIXED_LENGTH
RECORD_BYTES                    = 512
FILE_RECORDS                    = 6822
SEQUENCE_NUMBER                 = 7
END_OBJECT                      = FILE
END_OBJECT                      = DIRECTORY
```

A.12 DOCUMENT

The DOCUMENT object is used to label a particular document that is provided on a volume to support an archived data product. A document can be made up of one or more files in a single format. For instance, a document may be comprised of as many TIFF files as there are pages in the document.

Multiple versions of a document can be supplied on a volume with separate formats, requiring a DOCUMENT object for each document version (i.e., OBJECT = TEX_DOCUMENT and OBJECT = PS_DOCUMENT when including both the TEX and Postscript versions of the same document).

PDS requires that at least one version of any document be plain ASCII text in order to allow users the capability to read, browse, or search the text without requiring software or text processing packages. This version can be plain, unmarked text, or ASCII text containing a markup language. (See the Documentation chapter of this document for more details.)

The DOCUMENT object contains keywords that identify and describe the document, provide the date of publication of the document, indicate the number of files comprising the document, provide the format of the document files, and identify the software used to compress or encode the document, as applicable.

DOCUMENT labels must be detached files unless the files are plain, unmarked text that will not be read by text or word processing packages. A DOCUMENT object for each format type of a document can be included in the same label file with pointers, such as ^TIFF_DOCUMENT for a TIFF formatted document. (See example below.)

Required Keywords

1. DOCUMENT_NAME
2. DOCUMENT_TOPIC_TYPE
3. INTERCHANGE_FORMAT
4. DOCUMENT_FORMAT
5. PUBLICATION_DATE

Optional Keywords

1. ABSTRACT_TEXT
2. DESCRIPTION
3. ENCODING_TYPE
4. FILES

Required Objects

None

Optional Objects

None

Example

The following example detached label, PDSUG.LBL, is for a Document provided in three formats: ASCII text, TIFF, and TEX.

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                  = PDS3
RECORD_TYPE                     = UNDEFINED

^ASCII_DOCUMENT                 = "PDSUG.ASC"
^TIFF_DOCUMENT                  = { "PDSUG001.TIF", "PDSUG002.TIF",
                                    "PDSUG003.TIF", "PDSUG004.TIF" }
^TEX_DOCUMENT                   = "PDSUG.TEX"

OBJECT                          = ASCII_DOCUMENT
DOCUMENT_NAME                   = "Planetary Data System Data Set Catalog User's Guide"
PUBLICATION_DATE                = 1992-04-13
DOCUMENT_TOPIC_TYPE             = "USER'S GUIDE"
INTERCHANGE_FORMAT              = ASCII
DOCUMENT_FORMAT                 = TEXT

DESCRIPTION                     = "The Planetary Data System Data Set Catalog User's Guide describes the
fundamentals of accessing, searching, browsing, and ordering data from the PDS Data Set Catalog at the Central Node.  The text
for this 4-page document is provided here in this plain, ASCII text file."
ABSTRACT_TEXT                   = "The PDS Data Set Catalog is similar in function and purpose to a card catalog
in a library.  Use a Search screen to find data items, a List/Order screen to order data items, and the More menu option to see more
information."
END_OBJECT                      = ASCII_DOCUMENT

OBJECT                          = TIFF_DOCUMENT
DOCUMENT_NAME                   = "Planetary Data System Data Set Catalog User's Guide"
DOCUMENT_TOPIC_TYPE             = "USER'S GUIDE"
INTERCHANGE_FORMAT              =  BINARY
DOCUMENT_FORMAT                 =  TIFF
PUBLICATION_DATE                = 1992-04-13
FILES                           = 4
ENCODING_TYPE                   =  "CCITT/3"
DESCRIPTION                     = "
The Planetary Data System Data Set Catalog User's Guide describes the fundamentals of accessing, searching, browsing, and
ordering data from the PDS Data Set Catalog at the Central Node.
The 4-page document is provided here in 4 consecutive files, one file per page, in Tagged Image File Format (TIFF) using Group
3 compression.  It has been tested to successfully import into WordPerfect 5.0, FrameMaker, and Photoshop."
ABSTRACT_TEXT                   = "
The PDS Data Set Catalog is similar in function and purpose to a card catalog in a library.  Use a Search screen to find data items,
a List/Order screen to order data items, and the More menu option to see more information."
END_OBJECT                      = TIFF_DOCUMENT

OBJECT                          = TEX_DOCUMENT
DOCUMENT_NAME                   = "Planetary Data System Data Set Catalog User's Guide"
DOCUMENT_TOPIC_TYPE             = "USER'S GUIDE"
```

```
INTERCHANGE_FORMAT                  =  ASCII
DOCUMENT_FORMAT                     =  TEX
PUBLICATION_DATE                    = 1992-04-13
DESCRIPTION                         = "
The Planetary Data System Data Set Catalog User's Guide describes the fundamentals of accessing, searching, browsing, and
ordering data from the PDS Data Set Catalog at the Central Node.
The 4-page document is provided here in TeX format with all necessary macros included."
ABSTRACT_TEXT                       = "
The PDS Data Set Catalog is similar in function and purpose to a card catalog in a library.  Use a Search screen to find data items,
a List/Order screen to order data items, and the More menu option to see more information."
END_OBJECT                          = TEX_DOCUMENT
END
```

A.13 ELEMENT (Primitive Data Object)

The ELEMENT object provides a means of defining a lowest level component of a data object that is stored in an integral multiple of 8-bit bytes. Element objects may be embedded in COLLECTION and ARRAY data objects. The optional START_BYTE element identifies a location relative to the enclosing object. If not explicitly included, a START_BYTE = 1 is assumed for the ELEMENT.

Required Keywords

1. BYTES
2. DATA_TYPE
3. NAME

Optional Keywords

1. START_BYTE
2. BIT_MASK
3. DERIVED_MAXIMUM
4. DERIVED_MINIMUM
5. DESCRIPTION
6. FORMAT
7. INVALID_CONSTANT
8. MINIMUM
9. MAXIMUM
10. MISSING_CONSTANT
11. OFFSET
12. SCALING_FACTOR
13. UNIT
14. VALID_MINIMUM
15. VALID_MAXIMUM

Required Objects

None

Optional Objects

None

Example

Please refer to the example in the ARRAY Primitive object for an example of the implementation of the ELEMENT object.

A.14 FILE

The FILE object is used in attached or detached labels to define the attributes or characteristics of a data file. In attached labels, the file object is also used to indicate boundaries between label records and data records in data files which have attached labels. The FILE object may be used in three ways:

(1) As an implicit object in attached or detached labels. As depicted in the following example, all detached label files and attached labels contain an implicit FILE object which starts at the top of the label and ends where the label ends. In these cases, the PDS recommends against using the NAME keyword to reference the file name.

```
-----------------------------------------------------------------
     RECORD_TYPE       = FIXED_LENGTH
     RECORD_BYTES      = 80
     FILE_RECORDS      = 522
     LABEL_RECORDS     = 10
    (remainder of the label)
-----------------------------------------------------------------
```

For data products labelled using the implicit file object (e.g. for minimal labels) DATA_OBJECT_TYPE = FILE should be used in the Data Set Catalog Template.

(2) As an explicit object which is used when a file reference is needed in a combined detached or minimal label. In this case, the optional FILE_NAME element is used to identify the file being referenced.

```
-----------------------------------------------------------------
     OBJECT            = FILE
     FILE_NAME         = "IM10347.DAT"
     RECORD_TYPE       = STREAM
     FILE_RECORDS      = 1024
   (other optional keywords describing the file)
     END_OBJECT        = FILE
-----------------------------------------------------------------
```

For data products labelled using the explicit file object (e.g. for minimal labels) DATA_OBJECT_TYPE = FILE should be used in the Data Set Catalog Template.

(3) As an explicit object to identify specific files as sub-objects of the DIRECTORY in VOLUME objects. In this case, the optional FILE_NAME element is used to identify the file being referenced on a tape archive volume.

```
 -----------------------------------------------------------------
     OBJECT            = FILE
     FILE_NAME         = "VOLDESC.CAT"
     RECORD_TYPE       = STREAM
     SEQUENCE_NUMBER = 1
     END_OBJECT        = FILE
-----------------------------------------------------------------
```

The keywords in the FILE object always describe the file being referenced, and not the file in which the keywords are contained (i.e., if the FILE object is used in a detached label file, the FILE object keywords describe the detached data file, not the label file which contains the keywords). For example, if a detached label for a data file is being created and the label will be in STREAM format, but the data will be stored in a file having FIXED_LENGTH records, then the RECORD_TYPE keyword in the label file must be given the value FIXED_LENGTH.

The following table identifies data elements that are required (Req), optional (Opt), and not applicable (-) for various types of files

Labeling Method	Att	Det	Att	Det	Att	Det	Att	Det
RECORD_TYPE	FIXED_LENGTH		VARIABLE_LENGTH		STREAM		UNDEFINED	
RECORD_BYTES	Req	Req	Rmax	Rmax	Omax	-	-	-
FILE_RECORDS	Req	Req	Req	Req	Opt	Opt	-	-
LABEL_RECORDS	Req	-	Req	-	Opt	-	-	-

Required Keywords

1. RECORD_TYPE

(See above table for the conditions of use of additional required keywords)

Optional Keywords

1. FILE_NAME (required only in minimal detached labels and tape archives)
2. FILE_RECORDS (required only in minimal detached labels and tape archives)
3. LABEL_RECORDS
4. RECORD_BYTES
5. SEQUENCE_NUMBER

Required Objects

None

Optional Objects

None

Example

Below is an example of a set of explicit file objects in a combined detached label. An additional example of the use of explicit FILE object can be found in the VOLUME object.

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                          = PDS3
HARDWARE_MODEL_ID                       = "SUN SPARC STATION"
OPERATING_SYSTEM_ID                     = "SUN OS 4.1.1"
SPACECRAFT_NAME                         = "VOYAGER 2"
INSTRUMENT_NAME                         = "PLASMA WAVE RECEIVER"
MISSION_PHASE_NAME                      = "URANUS ENCOUNTER"
TARGET_NAME                             = URANUS
DATA_SET_ID                             = "VG2-U-PWS-4-RDR-SA-48.0SEC-V1.0"
PRODUCT_ID                              = "T860123-T860125"

OBJECT                                  = FILE
FILE_NAME                               = "T860123.DAT"
FILE_RECORDS                            = 1800
RECORD_TYPE                             = FIXED_LENGTH
RECORD_BYTES                            = 105
START_TIME                              = 1986-01-23T00:00:00.000Z
STOP_TIME                               = 1986-01-24T00:00:00.000Z
^TIME_SERIES                            = "T860123.DAT"

OBJECT                                  = TIME_SERIES
INTERCHANGE_FORMAT                      = BINARY
ROWS                                    = 1800
ROW_BYTES                               = 105
COLUMNS                                 = 19
^STRUCTURE                              = "PWS_DATA.FMT"
SAMPLING_PARAMETER_NAME                 = TIME
SAMPLING_PARAMETER_UNIT                 = SECOND
SAMPLING_PARAMETER_INTERVAL             = 48.0
END_OBJECT                              = TIME_SERIES
END_OBJECT                              = FILE

OBJECT                                  = FILE
FILE_NAME                               = "T860124.DAT"
FILE_RECORDS                            = 1800
RECORD_TYPE                             = FIXED_LENGTH
RECORD_BYTES                            = 105
START_TIME                              = 1986-01-24T00:00:00.000Z
STOP_TIME                               = 1986-01-25T00:00:00.000Z
^TIME_SERIES                            = "T860124.DAT"

OBJECT                                  = TIME_SERIES
INTERCHANGE_FORMAT                      = BINARY
ROWS                                    = 1800
ROW_BYTES                               = 105
COLUMNS                                 = 19
^STRUCTURE                              = "PWS_DATA.FMT"
SAMPLING_PARAMETER_NAME                 = TIME
SAMPLING_PARAMETER_UNIT                 = SECOND
SAMPLING_PARAMETER_INTERVAL             = 48.0
END_OBJECT                              = TIME_SERIES
END_OBJECT                              = FILE
```

```
OBJECT                             = FILE
FILE_NAME                          = "T860125.DAT"
FILE_RECORDS                       = 1799
RECORD_TYPE                        = FIXED_LENGTH
RECORD_BYTES                       = 105
START_TIME                         = 1986-01-30T00:00:00.000Z
STOP_TIME                          = 1986-01-30T23:59:12.000Z
^TIME_SERIES                       = "T860125.DAT"

OBJECT                             = TIME_SERIES
INTERCHANGE_FORMAT                 = BINARY
ROWS                               = 1799
ROW_BYTES                          = 105
COLUMNS                            = 19
^STRUCTURE                         = "PWS_DATA.FMT"
SAMPLING_PARAMETER_NAME            = TIME
SAMPLING_PARAMETER_UNIT            = SECOND
SAMPLING_PARAMETER_INTERVAL        = 48.0
END_OBJECT                         = TIME_SERIES
END_OBJECT                         = FILE

END
```

A.15 GAZETTEER_TABLE

The GAZETTEER_TABLE object is a specific type of a TABLE object that provides information about the geographical features of a planet or satellite. It contains information about a named feature such as location, size, origin of feature name, etc. The GAZETTEER_TABLE contains one row for each feature named on the target body. The table is formatted so that it may be read directly by many data management systems on various host computers. All fields (columns) are separated by commas, and character fields are enclosed by double quotation marks. Each record consist of 480 bytes, with a carriage return/line feed sequence in bytes 479 and 480. This allows the table to be treated as a fixed length record file on hosts that support this file type and as a normal text file on other hosts.

Currently the PDS Imaging Node at the USGS is the data producer for all GAZETTEER_TABLEs.

Required Keywords

1. NAME
2. INTERCHANGE_FORMAT
3. ROWS
4. COLUMNS
5. ROW_BYTES
6. DESCRIPTION

Optional Keywords

None

Required Objects

1. COLUMN

Optional Objects

None

Required COLUMN Objects (NAME =)

TARGET_NAME
SEARCH_FEATURE_NAME
DIACRITIC_FEATURE_NAME
MINIMUM_LATITUDE
MAXIMUM_LATITUDE
CENTER_LATITUDE
MINIMUM_LONGITUDE
MAXIMUM_LONGITUDE

CENTER_LONGITUDE
LABEL_POSITION_ID
FEATURE_LENGTH
PRIMARY_PARENTAGE_ID
SECONDARY_PARENTAGE_ID
MAP_SERIAL_ID
FEATURE_STATUS_TYPE
APPROVAL_DATE
FEATURE_TYPE
REFERENCE_NUMBER
MAP_CHART_ID
FEATURE_DESCRIPTION

Required Keywords (for Required COLUMN Objects)

NAME
DATA_TYPE
START_BYTE
BYTES
FORMAT
UNIT
DESCRIPTION

Example

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                  = PDS3
RECORD_TYPE                     = FIXED_LENGTH
RECORD_BYTES                    = 480
FILE_RECORDS                    = 1181
PRODUCT_ID                      =XYZ
TARGET_NAME                     = MARS
^GAZETTEER_TABLE                = "GAZETTER.TAB"

OBJECT                          = GAZETTEER_TABLE
NAME                            = "PLANETARY NOMENCLATURE GAZETTEER"
INTERCHANGE_FORMAT              = ASCII
ROWS                            = 1181
COLUMNS                         = 20
ROW_BYTES                       = 480
DESCRIPTION                     = "The gazetteer (file: GAZETTER.TAB) is a table of geographical features for
a planet or satellite.   It contains information about a named feature such as location, size, origin of feature name, etc. The Gazetteer
Table contains one row for each feature named on the target body.  The table is formatted so that it may be read directly into many
data management systems on various host computers. All fields (columns) are separated by commas, and character fields are
preceded by double quotation marks. Each record consist of 480 bytes, with a carriage return/line feed sequence in bytes 479 and
480. This allows the table to be treated as a fixed length record file on hosts that support this file type and as a normal text file on
other hosts."

OBJECT                          = COLUMN
NAME                            = TARGET_NAME
DATA_TYPE                       = CHARACTER
```

```
START_BYTE                              = 2
BYTES                                   = 20
FORMAT                                  = "A20"
UNIT                                    = "N/A"
DESCRIPTION                             = "The planet or satellite on which the feature is located."
END_OBJECT                              = COLUMN


OBJECT                                  = COLUMN
NAME                                    = SEARCH_FEATURE_NAME
DATA_TYPE                               = CHARACTER
START_BYTE                              = 25
BYTES                                   = 50
FORMAT                                  = "A50"
UNIT                                    = "N/A"
DESCRIPTION                             = "The geographical feature name with all diacritical marks stripped off.  This
name is stored in upper case only so that it can be used for sorting and search purposes. This field should not be used to designate
the name of the feature because it does not contain the diacritical marks. Feature names not containing diacritical marks can often
take on a completely different meaning and in some cases the meaning can be deeply offensive."
END_OBJECT                              = COLUMN

OBJECT                                  = COLUMN
NAME                                    = DIACRITIC_FEATURE_NAME
DATA_TYPE                               = CHARACTER
START_BYTE                              = 78
BYTES                                   = 100
FORMAT                                  = "A100"
UNIT                                    = "N/A"
DESCRIPTION                             = "The geographical feature name containing standard diacritical information.  A
detailed description of the diacritical mark formats are described in  the gazetteer documentation.
```

DIACRITICALS USED IN THE TABLE

The word diacritic comes from a Greek word meaning to separate. It refers to the accent marks employed to separate, or distinguish, one form of pronunciation of a vowel or consonant from another.

This note is included to familiarize the user with the codes used to represent diacriticals found in the table, and the values usually associated with them. In the table, the code for a diacritical is preceded by a backslash and is followed, without a space, by the letter it is modifying.

This note is organized as follows: the code is listed first, followed by the name of the accent mark, if applicable, a brief description of the appearance of the diacritical and a short narrative on its usage.

acute accent; a straight diagonal line extending from upper right to lower left. The acute accent is used in most languages to lengthen a vowel; in some, such as Oscan, to denote an open vowel. The acute is also often used to indicate the stressed syllable; in some transcriptions it indicates a palatalized consonant.

diaeresis or umlaut; two dots surmounting the letter. In Romance languages and English, the diaeresis is used to indicate that consecutive vowels do not form a dipthong (see below); in modern German and Scandinavian languages, it denotes palatalization of vowels.

circumflex; a chevron or inverted 'v' shape, with the apex at the top. Used most often in modern languages to indicate lengthening of a vowel.

tilde; a curving or waving line above the letter. The tilde is a form of circumflex. The tilde is used most often in Spanish to form a palatalized n as in the word 'ano', pronounced 'anyo'. It is also used occasionally to indicate nasalized vowels.

macron; a straight line above the letter. The macron is used almost universally to lengthen a vowel.

breve; a concave semicircle or 'u' shape surmounting the letter. Originally used in Greek, the breve indicates a short vowel.

a small circle or 'o' above the letter. Frequently used in Scandinavian languages to indicate a broad 'o'.

e dipthong or ligature; transcribed as two letters in contact with each other. The dipthong is a combination of vowels that are pronounced together.

cedilla; a curved line surmounted by a vertical line, placed at the bottom of the letter. The cedilla is used in Spanish and French to denote a dental, or soft, 'c'. In the new Turkish transcription, 'c' cedilla has the value of English 'ch'. In Semitic languages, the cedilla under a consonant indicates that it is emphatic.

check or inverted circumflex; a 'v' shape above the letter. This accent is used widely in Slavic languages to indicate a palatal articulation, like the consonant sounds in the English words chapter and shoe and the 'zh' sound in pleasure.

a single dot above the letter. This diacritical denotes various things; in Lithuanian, it indicates a close long vowel. In Sanskrit, when used with 'n', it is a velar sound, as in the English 'sink'; in Irish orthography, it indicates a fricative consonant (see below).

accent grave; a diagonal line (above the letter) extending from upper left to lower right. The grave accent is used in French, Spanish and Italian to denote open vowels.

fricative; a horizontal line through a consonant. A fricative consonant is characterized by a frictional rustling of the breath as it is emitted."

```
END_OBJECT                = COLUMN

OBJECT                    = COLUMN
NAME                      = MINIMUM_LATITUDE
DATA_TYPE                 = REAL
START_BYTE                = 180
BYTES                     = 7
FORMAT                     = "F7.2"
UNIT                      = DEGREE
DESCRIPTION               = "The minimum_latitude element specifies the southernmost latitude of a spatial
area, such as a map, mosaic, bin, feature, or region."
END_OBJECT                = COLUMN

OBJECT                    = COLUMN
NAME                      = MAXIMUM_LATITUDE
DATA_TYPE                 = REAL
START_BYTE                = 188
BYTES                     = 7
FORMAT                    = "F7.2"
UNIT                      = DEGREE
DESCRIPTION               = "The maximum_latitude element  specifies the northernmost latitude of a
spatial area, such as a map, mosaic, bin, feature, or region."
END_OBJECT                = COLUMN

OBJECT                    = COLUMN
NAME                      = CENTER_LATITUDE
DATA_TYPE                 = REAL
START_BYTE                = 196
BYTES                     = 7
FORMAT                    = "F7.2"
UNIT                      = DEGREE
DESCRIPTION               = "The center latitude of the feature."
END_OBJECT                = COLUMN

OBJECT                    = COLUMN
NAME                      = MINIMUM_LONGITUDE
DATA_TYPE                 = REAL
```

```
START_BYTE                              = 204
BYTES                                   = 7
FORMAT                                  = "F7.2"
UNIT                                    = DEGREE
DESCRIPTION                             = "The minimum_longitude element  specifies the easternmost latitude of a
spatial area, such as  a map, mosaic, bin, feature, or region.  "
END_OBJECT                              = COLUMN

OBJECT                                  = COLUMN
NAME                                    = MAXIMUM_LONGITUDE
DATA_TYPE                               = REAL
START_BYTE                              = 212
BYTES                                   = 7
FORMAT                                  = "F7.2"
UNIT                                    = DEGREE
DESCRIPTION                             = "The maximum_longitude element specifies the westernmost longitude of a
spatial area, such  as a map, mosaic, bin, feature, or region. "
END_OBJECT                              = COLUMN

OBJECT                                  = COLUMN
NAME                                    = CENTER_LONGITUDE
DATA_TYPE                               = REAL
START_BYTE                              = 220
BYTES                                   = 7
FORMAT                                  = "F7.2"
UNIT                                    = DEGREE
DESCRIPTION                             = "The center longitude of the feature."
  END_OBJECT                            = COLUMN

OBJECT                                  = COLUMN
NAME                                    = LABEL_POSITION_ID
DATA_TYPE                               = CHARACTER
START_BYTE                              = 229
BYTES                                   = 2
FORMAT                                  = "A2"
UNIT                                    = "N/A"
DESCRIPTION                             = "The suggested plotting position of the feature name (UL=Upper left,
UC=Upper center, UR=Upper right, CL=Center left, CR=Center right, LL=Lower left,  LC=Lower center, LR=Lower right). This
field is used to instruct the plotter where to place the typographical label with respect to the center of the feature.  This code is used
to avoid crowding of names in areas where there is a high density of named features."
END_OBJECT                              = COLUMN

OBJECT                                  = COLUMN
NAME                                    = FEATURE_LENGTH
DATA_TYPE                               = REAL
START_BYTE                              = 233
BYTES                                   = 8
FORMAT                                  = "F8.2"
UNIT                                    = KILOMETER
DESCRIPTION                             = "The longer or longest dimension  of an object. For the Gazetteer usage, this
field refers to the length of the named feature."
END_OBJECT                              = COLUMN

OBJECT                                  = COLUMN
NAME                                    = PRIMARY_PARENTAGE_ID
DATA_TYPE                               = CHARACTER
START_BYTE                              = 243
BYTES                                   = 2
FORMAT                                  = "A2"
```

```
UNIT                          = "N/A"
DESCRIPTION                   = "This field contains the primary  origin of the feature name (i.e. where the name
originated).   It contains a code for the continent or country origin of the name. Please see Appendix 5 of the gazetteer documentation
(GAZETTER.TXT) for a definition of the codes used to define the continent or country."
END_OBJECT                    = COLUMN

OBJECT                        = COLUMN
NAME                          = SECONDARY_PARENTAGE_ID
DATA_TYPE                     = CHARACTER
START_BYTE                    = 248
BYTES                         = 2
FORMAT                        = "A2"
UNIT                          = "N/A"
DESCRIPTION                   = "This field contains the secondary origin of the feature name. It contains a code
for a country, state, territory, or ethnic group. Please see  Appendix 5 of the gazetteer documentation (GAZETTER.TXT) for  a
defintion of the codes in this field."

END_OBJECT                    = COLUMN

OBJECT                        = COLUMN
NAME                          = MAP_SERIAL_ID
DATA_TYPE                     = CHARACTER
START_BYTE                    = 253
BYTES                         = 6
FORMAT                        = "A6"
UNIT                          = "N/A"
DESCRIPTION                   = "The identification of the map that contains the named feature. This field
represents the map serial number of the map publication used for ordering maps from the U.S. Geological Survey. The map
identified in this field best portrays the named feature."
  END_OBJECT                  = COLUMN

OBJECT                        = COLUMN
NAME                          = FEATURE_STATUS_TYPE
DATA_TYPE                     = CHARACTER
START_BYTE                    = 262
BYTES                         = 12
FORMAT                        = "A12"
UNIT                          = "N/A"
DESCRIPTION                   = "The IAU approval status of the named feature. Permitted values are
'PROPOSED', 'PROVISIONAL', 'IAU-APPROVED', and 'DROPPED'. Dropped names have been disallowed by the IAU.
However, these features have been included in the gazetteer for historical purposes. Some named features that are disallowed by the
IAU may commonly be used on some maps."
END_OBJECT                    = COLUMN

OBJECT                        = COLUMN
NAME                          = APPROVAL_DATE
DATA_TYPE                     = INTEGER
START_BYTE                    = 276
BYTES                         = 4
FORMAT                        = "I4"
UNIT                          = "N/A"
DESCRIPTION                   = "Date at which an object has been approved by the officially sanctioned
organization. This field contains the year the IAU approved the feature name."
END_OBJECT                    = COLUMN

OBJECT                        = COLUMN
NAME                          = FEATURE_TYPE
DATA_TYPE                     = CHARACTER
START_BYTE                    = 282
```

BYTES = 20
FORMAT = "A20"
UNIT = "N/A"
DESCRIPTION = "The feature type identifies the type of a particular feature, according to IAU standards. Examples are 'CRATER', 'TESSERA', 'TERRA', etc. See Appendix 7 of the gazetteer documentation (GAZETTER.TXT).
DESCRIPTOR TERMS (FEATURE TYPES)

FEATURE	DESCRIPTION
ALBEDO FEATURE	Albedo feature
CATENA	Chain of craters
CAVUS	Hollows, irregular depressions
CHAOS	Distinctive area of broken terrain
CHASMA	Canyon
COLLES	Small hill or knob
CORONA	Ovoid-shaped feature
CRATER	Crater
DORSUM	Ridge
ERUPTIVE CENTER	Eruptive center
FACULA	Bright spot
FLEXUS	Cuspate linear feature
FLUCTUS	Flow terrain
FOSSA	Long, narrow, shallow depression
LABES	Landslide
LABYRINTHUS	Intersecting valley complex
LACUS	Lake
LARGE RINGED FEATURE	Large ringed feature
LINEA	Elongate marking
MACULA	Dark spot
MARE	Sea
MENSA	Mesa, flat-topped elevation
MONS	Mountain
OCEANUS	Ocean
PALUS	Swamp
PATERA	Shallow crater; scalloped, complex edge
PLANITIA	Low plain
PLANUM	Plateau or high plain
PROMONTORIUM	Cape
REGIO	Region
RIMA	Fissure
RUPES	Scarp

```
SCOPULUS                          Lobate or irregular scarp
SINUS                             Bay
SULCUS                            Subparallel furrows and ridges
TERRA                             Extensive land mass
TESSERA                           Tile; polygonal ground
THOLUS                            Small domical mountain or hill
UNDAE                             Dunes
VALLIS                            Sinuous valley
VASTITAS                          Widespread lowlands
VARIABLE FEATURE                  Variable feature "

END_OBJECT                    = COLUMN

OBJECT                        = COLUMN
NAME                          = REFERENCE_NUMBER
DATA_TYPE                     = INTEGER
START_BYTE                    = 304
BYTES                         = 4
FORMAT                        = "I4"
UNIT                          = "N/A"
DESCRIPTION                   = "Literature reference from which the spelling and description of the feature
name was derived.  See Appendix 6 of the gazetteer documentation  (GAZETTER.TXT)."
END_OBJECT                    = COLUMN

OBJECT                        = COLUMN
NAME                          = MAP_CHART_ID
DATA_TYPE                     = CHARACTER
START_BYTE                    = 310
BYTES                         = 6
FORMAT                        = "A6"
UNIT                          = "N/A"
DESCRIPTION                   = "This field contains the abbreviation of the map designator or chart
identification (example MC-19, MC-18, etc.)."
END_OBJECT                    = COLUMN

OBJECT                        = COLUMN
NAME                          = FEATURE_DESCRIPTION
DATA_TYPE                     = CHARACTER
START_BYTE                    = 319
BYTES                         = 159
FORMAT                        = "A159"
UNIT                          = "N/A"
DESCRIPTION                   = "Short description of the feature name."
END_OBJECT                    = COLUMN
END_OBJECT                    = GAZETTEER_TABLE

END
```

A.16 HEADER

The HEADER object is used to identify and define the attributes of commonly used header data structures such as VICAR or FITS. These structures are usually system or software specific and are described in detail in a referenced description text file. The use of bytes within the header object refers to the number of bytes for the entire header, not a single record.

Required Keywords

1. BYTES
2. HEADER_TYPE

Optional Keywords

1. DESCRIPTION
2. INTERCHANGE_FORMAT
3. RECORDS

Required Objects

None

Optional Objects

None

Example

The following example shows the detached label file "TIMTC02A.LBL". The label describes the data product file "TIMTC02A.IMG" which contains a HEADER object followed by an IMAGE object.

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                    = PDS3
/* PDS label for a TIMS image */
RECORD_TYPE                       = FIXED_LENGTH
RECORD_BYTES                      = 638
FILE_RECORDS                      = 39277
/* Pointers to objects */
^IMAGE_HEADER                     = ("TIMTC02A.IMG",1)
^IMAGE                            = ("TIMTC02A.IMG",2)
/* Image description */
DATA_SET_ID                       = "C130-E-TIMS-2-EDR-IMAGE-V1.0"
PRODUCT_ID                        = "TIMTC02A"
INSTRUMENT_HOST_NAME              = "NASA C-130 AIRCRAFT"
INSTRUMENT_NAME                   = "THERMAL INFRARED MULTISPECTRAL SCANNER"
TARGET_NAME                       = EARTH
FEATURE_NAME                      = "TRAIL CANYON FAN"
START_TIME                        = 1989-09-29T21:47:35Z
```

```
STOP_TIME                          = 1989-09-29T21:47:35Z
CENTER_LATITUDE                    = 36.38
CENTER_LONGITUDE                   = 116.96
INCIDENCE_ANGLE                    = 0.0
EMISSION_ANGLE                     = 0.0
/* Description of objects */
OBJECT                             = IMAGE_HEADER
BYTES                              = 638
RECORDS                            = 1
HEADER_TYPE                        = VICAR2
INTERCHANGE_FORMAT                 = BINARY
^DESCRIPTION                       = "VICAR2.TXT"
END_OBJECT                         = IMAGE_HEADER

OBJECT                             = IMAGE
LINES                              = 6546
LINE_SAMPLES                       = 638
SAMPLE_TYPE                        = UNSIGNED_INTEGER
SAMPLE_BITS                        = 8
SAMPLE_BIT_MASK                    = 2#11111111#
BANDS                              = 6
BAND_STORAGE_TYPE                  = LINE_INTERLEAVED
END_OBJECT                         = IMAGE
END
```

A.17 HISTOGRAM

The HISTOGRAM object is a sequence of numeric values that provides the number of occurrences of a data value or a range of data values in a data object. The number of items in a histogram will normally be equal to the number of distinct values allowed in a field of the data object. For example, an 8 bit integer field can have a maximum of 256 values, and would result in a 256 item histogram. Histograms may be used to bin data, in which case an offset and scaling factor indicate the dynamic range of the data represented.

The following equation allows the calculation of the range of each 'bin' in the histogram.

'bin lower boundary' = 'bin element' * scale_factor + offset

Required Keywords

1. ITEMS
2. DATA_TYPE
3. ITEM_BYTES

Optional Keywords

1. BYTES
2. INTERCHANGE_FORMAT
3. OFFSET
4. SCALING_FACTOR

Required Objects

None

Optional Objects

None

Example

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                         = PDS3
/*      FILE FORMAT AND LENGTH */

RECORD_TYPE                            = FIXED_LENGTH
RECORD_BYTES                           = 956
FILE_RECORDS                           = 965
LABEL_RECORDS                          = 3

/*      POINTERS TO START RECORDS OF OBJECTS IN FILE */

^IMAGE_HISTOGRAM                       = 4
```

```
^IMAGE                                  = 6

/*      IMAGE DESCRIPTION */

DATA_SET_ID                             = "VO1/VO2-M-VIS-5-DIM-V1.0"
PRODUCT_ID                              = "MG15N022-GRN-666A"
SPACECRAFT_NAME                         = VIKING_ORBITER_1
TARGET_NAME                             = MARS
START_TIME                              = 1978-01-14T02:00:00
STOP_TIME                               = 1978-01-14T02:00:00
SPACECRAFT_CLOCK_START_TIME             = UNK
SPACECRAFT_CLOCK_STOP_TIME              = UNK
PRODUCT_CREATION_TIME                   = 1995-01-01T00:00:00
ORBIT_NUMBER                            = 666
FILTER_NAME                             = GREEN
IMAGE_ID                                = "MG15N022-GRN-666A"
INSTRUMENT_NAME                         = {VISUAL_IMAGING_SUBSYSTEM_CAMERA_A,
VISUAL_IMAGING_SUBSYSTEM_CAMERA_B}
NOTE                                    = "MARS MULTI-SPECTRAL MDIM SERIES"
/* SUN RAYS EMISSION, INCIDENCE, AND PHASE ANGLES OF IMAGE CENTER*/
SOURCE_PRODUCT_ID                        =" 666A36"
EMISSION_ANGLE                          = 21.794
INCIDENCE_ANGLE                         = 66.443
PHASE_ANGLE                             = 46.111

/*      DESCRIPTION OF OBJECTS CONTAINED IN FILE */

OBJECT                                  = IMAGE_HISTOGRAM
ITEMS                                   = 256
DATA_TYPE                               = VAX_INTEGER
ITEM_BYTES                              = 4
END_OBJECT                              = IMAGE_HISTOGRAM

OBJECT                                  = IMAGE
LINES                                   = 960
LINE_SAMPLES                            = 956
SAMPLE_TYPE                             = UNSIGNED_INTEGER
SAMPLE_BITS                             = 8
SAMPLE_BIT_MASK                         = 2#11111111#
CHECKSUM                                = 65718982
/*  I/F = SCALING_FACTOR*DN + OFFSET, CONVERT TO INTENSITY/FLUX */
SCALING_FACTOR                          = 0.001000
OFFSET                                  = 0.0
/* OPTIMUM COLOR STRETCH FOR DISPLAY OF COLOR IMAGES */
STRETCHED_FLAG                          = FALSE
STRETCH_MINIMUM                         = ( 53,  0)
STRETCH_MAXIMUM                         = (133,255)
END_OBJECT                              = IMAGE

END
```

A.18 HISTORY

A HISTORY object is a dynamic description of the history of one or more associated data objects in a file. It supplements the essentially static description contained in the PDS label.

The HISTORY object contains text in a format similar to that of the ODL statements used in the label. It identifies previous computer manipulation of the principal data object(s) in the file. It includes an identification of the source data, processes performed, processing parameters, as well as dates and times of processing. It is intended that the history be available for display, be dynamically extended by any process operating on the data, and automatically propagated to the resulting data file. Eventually, it might be extracted for loading in detailed level catalogs of data set contents.

The HISTORY object is structured as a series of History Entries, one for each process which has operated on the data. Each entry contains a standard set of ODL element assignment statements, delimited by GROUP = program_name and END_GROUP = program_name statements. A subgroup in each entry, delimited by GROUP = PARAMETERS and END_GROUP = PARAMETERS, contains statements specifying the values of all parameters of the program.

HISTORY ENTRY ELEMENTS

Attribute	Description
VERSION_DATE	Program version date, ISO standard format.
DATE_TIME	Run date and time, ISO standard format.
NODE_NAME	Network name of computer.
USER_NAME	Username.
SOFTWARE_DESC	Program-generated (brief) description.
USER_NOTE	User-supplied (brief) description.

Unlike the above elements, the names of the parameters defined in the PARAMETERS subgroup are uncontrolled, and must only conform to the program.

The last entry in a HISTORY object is followed by an END statement. The HISTORY object, by convention, follows the PDS label of the file, beginning on a record boundary, and is located by a pointer statement in the label. There are no required elements for the PDS label description of the object; it is represented in the label only by the pointer statement, and OBJECT = HISTORY and END_OBJECT = HISTORY statements.

The HISTORY capability has been implemented as part of the Integrated Software for Imaging Spectrometers (ISIS) system (see QUBE object definition). ISIS Qube applications add their own entries to the Qube file's cumulative History object. ISIS programs run under NASA's TAE (Transportable Applications Executive) system, and are able to automatically insert all parameters of their TAE procedure into the history entry created by the program. Consult the ISIS System Design document for details and limitations imposed by that system. (See the QUBE object description for further references.)

Required Keywords

None

Optional Keywords

None

Required Objects

None

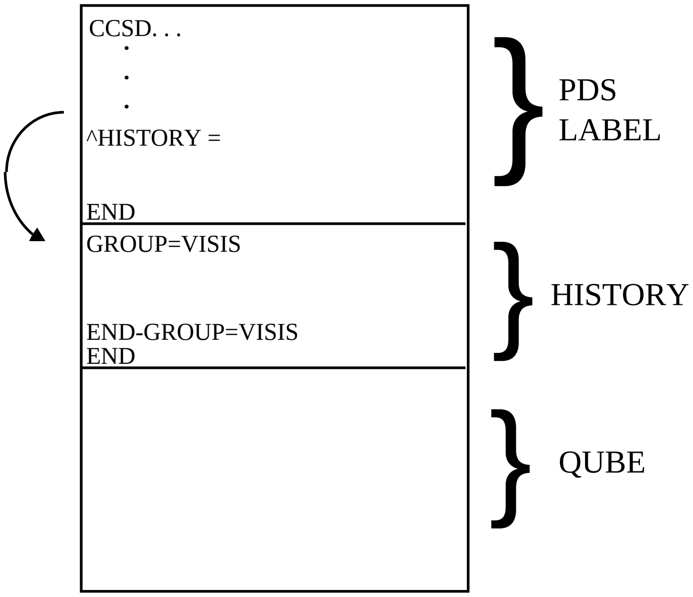
Optional Objects

None

Example

The following single-entry HISTORY object is from a Vicar-generated PDS-labeled qube file. (See the Qube object example.) There's only one entry because the qube (or rather its label) was generated by a single program, VISIS. A qube generated by multiple ISIS programs would have multiple history entries, represented by multiple GROUPs in the HISTORY object.

This diagram illustrates the placement of the example HISTORY object within a Qube data product with an attached PDS label.



```
GROUP                                  = VISIS
  VERSION_DATE                         = 1990-11-08
  DATE_TIME                              = 1991-07-25T10:12:52
  SOFTWARE_DESC                          = "ISIS cube file with PDS label has been generated as systematic product by
MIPL using the following programs:
      NIMSMERGE to create EDR's;
      NIMSCMM to create the merged mosaic & geometry cube;
      HIST2D to create a two-dimensional histogram;
      SPECPLOT to create the spectral plots;
      TRAN, F2, and INSERT3D to create the SII cube;
      VISIS to create the ISIS cube."

  USER_NOTE                              = "VPDIN1/ Footprint, Limbfit, Height=50"

  GROUP                                  = PARAMETERS
   EDR_FILE_NAME                         = " "          /*EDR accessed through MIPL Catalog*/
    IMAGE_ID                             = NULL
    SPICE_FILE_NAME                      = " "
    SPIKE_FILE_NAME                      = "mipl:[mipl.gll]boom_obscuration.nim"
    DARK_VALUE_FILE_NAME                 = " "
    CALIBRATION_FILE_NAME                = "ndat:nimsgs2.cal"
    MERGED_MOSAIC_FILE_NAME              = "ndat:vpdin1_dn_fp_lf_h50.CUB"
    DARK_INTERPOLATION_TYPE              = NOUPDAT
    PHOTOMETRIC_CORRECTION_TYPE          = NONE
    CUBE_NIMSEL_TYPE                     = NOCAL
    BINNING_TYPE                         = FOOTPRNT
    FILL_BOX_SIZE                        = 0
    FILL_MIN_VALID_PIXELS                = 0
    SUMMARY_IMAGE_RED_ID                 = 0
    SUMMARY_IMAGE_GREEN_ID               = 0
    SUMMARY_IMAGE_BLUE_ID                = 0
    ADAPT_STRETCH_SAT_FRAC               = 0.000000
    ADAPT_STRETCH_SAMP_FRAC              = 0.000000
    RED_STRETCH_RANGE                    = (    0,    0)
    GREEN_STRETCH_RANGE                  = (    0,    0)
    BLUE_STRETCH_RANGE                   = (    0,    0)
  END_GROUP                              = PARAMETERS
 END_GROUP                               = VISIS
 END
```

A.19 IMAGE

An IMAGE object is an array of sample values. Image objects are normally processed with special display tools to produce a visual representation of the sample values. This is done by assigning brightness levels or display colors to the various sample values. Images are composed of LINES and SAMPLES. They may contain multiple bands, in one of several storage orders.

Simple IMAGE objects are defined as having LINES as the number of horizontal lines, with each line having LINE_SAMPLES as the number of sample values defined. The default sample values are 8-bit unsigned binary integer. The sample size can be over- ridden using the SAMPLE_BITS keyword (e.g. SAMPLE_BITS = 32). The SAMPLE_TYPE keyword can be used to override the default SAMPLE_TYPE (e.g. SAMPLE_TYPE = VAX_REAL).

Each line of an IMAGE object may also be organized with a set of PREFIX or SUFFIX bytes, which provide engineering parameters related to each line. The PREFIX or SUFFIX area is treated as a TABLE object which has been concatenated with the IMAGE object. Each physical record in the file contains a row of the PREFIX or SUFFIX table and a line of the IMAGE. While this is a commonly used format for IMAGE storage, it can cause difficulties if used with general purpose display and processing software. In particular, most programs will consider the PREFIX and SUFFIX as part of the image, meaning that statistics generated for the image (mean, standard deviation, etc.) will be in error. It is recommended that PREFIX or SUFFIX information be stored as a separate TABLE data object in separate records within the file and not concatenated with the image data. (See Figure A.1.)

Most images are composed of LINES containing a horizontal array of SAMPLES. However some imaging sensors may scan in a vertical direction, creating an array of vertical lines, as in the case of the Viking Lander camera system.

More complex IMAGE formats include multi-band images, where SAMPLES or LINES of the same scene from several spectral bands are combined in one object, by sample (SAMPLE_INTERLEAVED), or by line (LINE_INTERLEAVED). Another IMAGE format is TILED, where a large IMAGE is divided into smaller pieces (TILES) to provide efficient access.

Figure A.2 illustrates the BANDS, BAND_NAME, and BAND_STORAGE_TYPE keywords that can be used to describe multi-band images.

Note: Additional engineering values may be prepended or appended to each LINE of an image, and are stored as concatenated TABLE objects, which must be named LINE_PREFIX and LINE_SUFFIX. IMAGE objects may be associated with other objects, including HISTOGRAMs, PALETTEs, HISTORY and TABLEs which contain statistics, display parameters, engineering values or other ancillary data.

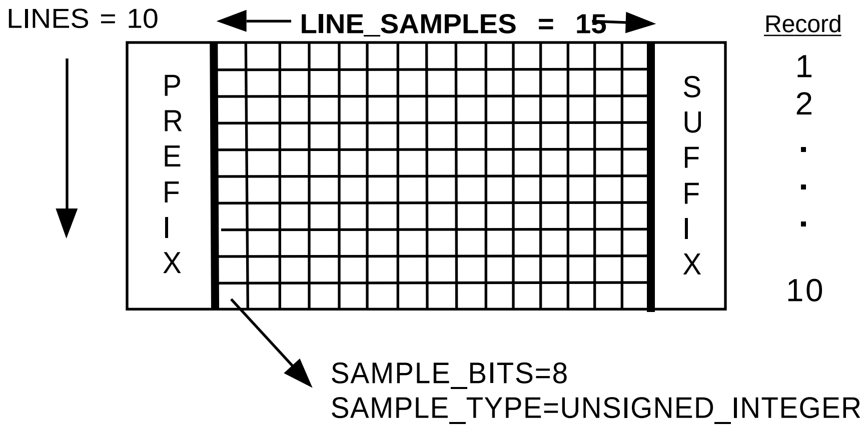


Figure A.1: Prefix and Suffix Bytes attached to an Image

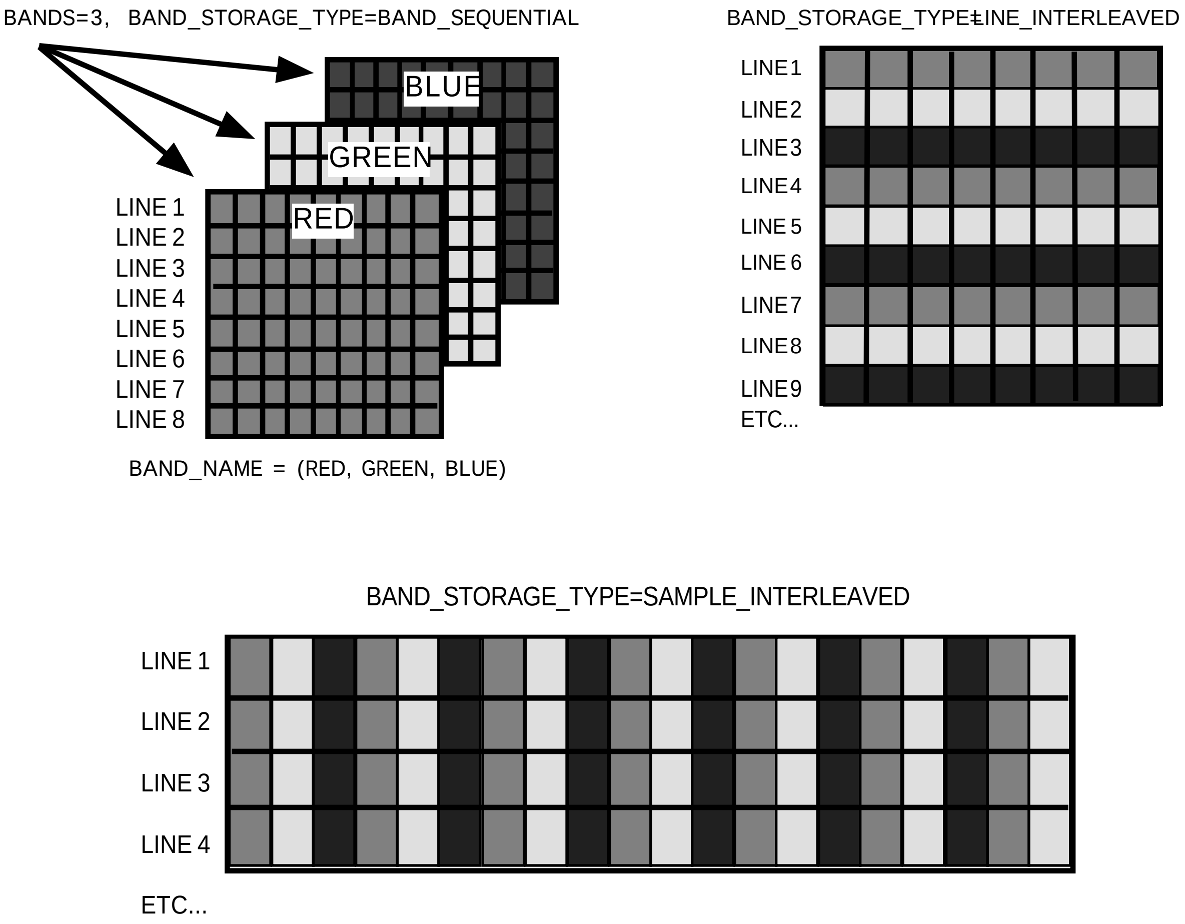


Figure A.2: Keywords for a Multi-Band Image

Required Keywords

1. LINES
2. LINE_SAMPLES
3. SAMPLE_TYPE
4. SAMPLE_BITS

Optional Keywords

1. BAND_SEQUENCE
2. BAND_STORAGE_TYPE
3. BANDS
4. CHECKSUM
5. DERIVED_MAXIMUM
6. DERIVED_MINIMUM
7. DESCRIPTION
8. ENCODING_TYPE
9. FIRST_LINE
10. FIRST_LINE_SAMPLE
11. INVALID_CONSTANT
12. LINE_PREFIX_BYTES
13. LINE_SUFFIX_BYTES
14. MISSING _CONSTANT
15. OFFSET
16. SAMPLE_BIT_MASK
17. SAMPLING_FACTOR
18. SCALING_FACTOR
19. SOURCE_FILE_NAME
20. SOURCE_LINES
21. SOURCE_LINE_SAMPLES
22. SOURCE_SAMPLE_BITS
23. STRETCHED_FLAG
24. STRETCH_MINIMUM
25. STRETCH_MAXIMUM

Required Objects

None

Optional Objects

None

Example

This is an example of an attached IMAGE label for a color digital mosaic image from the Mars Digital Image Map CD-ROMs. It includes a CHECKSUM to support automated volume production and validation, a SCALING_FACTOR to indicate the relationship between sample values and geophysical parameters and stretch keywords to indicate optimal values for image display.

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                      = PDS3

/*  FILE FORMAT AND LENGTH */

RECORD_TYPE                         = FIXED_LENGTH
RECORD_BYTES                        = 956
FILE_RECORDS                        = 965
LABEL_RECORDS                       = 3

/*  POINTERS TO START RECORDS OF OBJECTS IN FILE */

^IMAGE_HISTOGRAM                    = 4
^IMAGE                              = 6

/*  IMAGE DESCRIPTION */

DATA_SET_ID                         = "VO1/VO2-M-VIS-5-DIM-V1.0"
PRODUCT_ID                          = "MG15N022-GRN-666A"
SPACECRAFT_NAME                     = VIKING_ORBITER_1
TARGET_NAME                         = MARS
IMAGE_TIME                          = 1978-01-14T02:00:00
START_TIME                          = UNK
STOP_TIME                           = UNK
SPACECRAFT_CLOCK_START_COUNT        = UNK
SPACECRAFT_CLOCK_STOP_COUNT         = UNK
PRODUCT_CREATION_TIME               = 1995-01-01T00:00:00
ORBIT_NUMBER                        = 666
FILTER_NAME                         = GREEN
IMAGE_ID                            = "MG15N022-GRN-666A"
INSTRUMENT_NAME                     = {VISUAL_IMAGING_SUBSYSTEM_CAMERA_A,
                                      VISUAL_IMAGING_SUBSYSTEM_CAMERA_B}
NOTE                                = "MARS MULTI-SPECTRAL MDIM SERIES"
SOURCE_PRODUCT_ID                   = "666A36"
EMISSION_ANGLE                      = 21.794
INCIDENCE_ANGLE                     = 66.443
PHASE_ANGLE                         = 46.111

/*  DESCRIPTION OF OBJECTS CONTAINED IN FILE */

OBJECT                              = IMAGE_HISTOGRAM
ITEMS                               = 256
DATA_TYPE                           = VAX_INTEGER
ITEM_BYTES                          = 4
END_OBJECT                          = IMAGE_HISTOGRAM
```

```
OBJECT                          = IMAGE
LINES                           = 960
LINE_SAMPLES                    = 956
SAMPLE_TYPE                     = UNSIGNED_INTEGER
SAMPLE_BITS                     = 8
SAMPLE_BIT_MASK                 = 2#11111111#
CHECKSUM                        = 65718982
SCALING_FACTOR                  = 0.001000    /* I/F = scaling factor * DN + offset, */
                                 /* convert to intensity/flux.        */
OFFSET                          = 0.0
STRETCHED_FLAG                  = FALSE       /* Optimum color stretch for display   */
STRETCH_MINIMUM                 = ( 53,  0)   /* of color images.                    */
STRETCH_MAXIMUM                 = (133,255)
END_OBJECT                      = IMAGE

END
```

A.20 IMAGE MAP PROJECTION

The IMAGE_MAP_PROJECTION object is one of two distinct objects that define the map projection used in creating the digital images in a PDS data set.The name of the other associated object that completes the definition is called DATA_SET_MAP_PROJECTION.(see Appendix B)

The map projection information resides in these two objects, essentially to reduce data redundancy and at the same time allow the inclusion of elements needed to process the data at the image level. Basically, static information that is applicable to the complete data set reside in the DATA_SET_MAP_PROJECTION object, while dynamic information that is applicable to the individual images reside in the IMAGE_MAP_PROJECTION object.

The line_first_pixel, line_last_pixel, sample_first_pixel, and sample_last_pixel keywords are used to indicate which way is up in an image. Sometimes an image can be shifted or flipped prior to it being physically recorded. These keywords are used in calculating the mapping of pixels between the original image and the stored image.

The following equations give the byte offsets needed to determine the mapping of a pixel (X,Y) from the original image to a pixel in the stored image:

The sample offset from the first pixel is:

$$\frac{\text{sample\_bits} * (Y - \text{sample\_first\_pixel}) * \text{line\_samples}}{8 * (\text{sample\_last\_pixel} - \text{sample\_first\_pixel} + 1)}$$

The line offset from the first image line is:

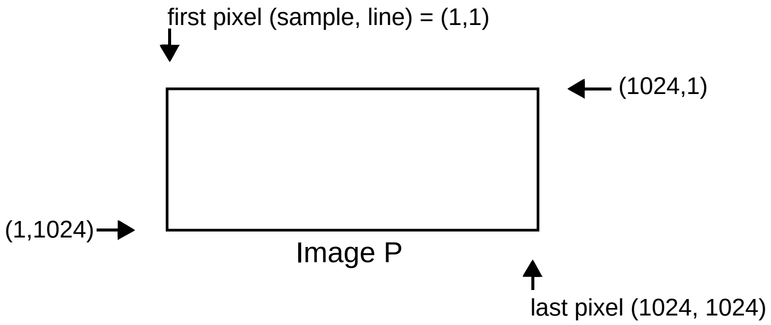
$$\frac{(X - \text{line\_first\_pixel}) * \text{lines}}{(\text{line\_last\_pixel} - \text{line\_first\_pixel} + 1)}$$

Additionally, in any image, ABS (sample_last_pixel - sample_first_pixel + 1) is always equal to line_samples, and ABS (line_last_pixel - line_first_pixel + 1) is always equal to lines.

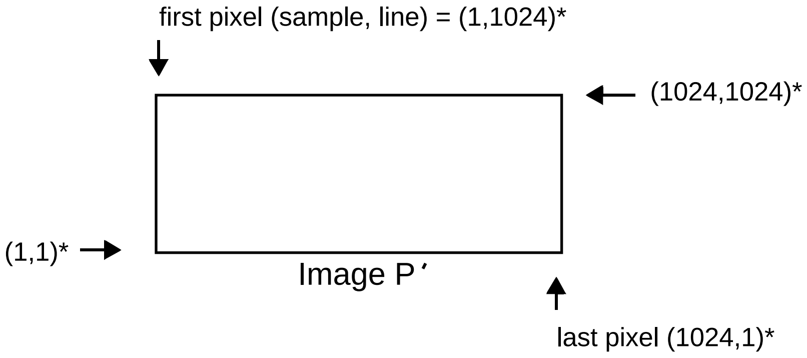
Example

Take a 1K by 1K 8-bit image which is rotated about the x-axis 180 degrees prior to being physically recorded.

Original Image: Positive direction is to the right and down



Stored Image: Positive direction is to the right and up



These pixel location values (*) are the positions from the original image. For example, the first pixel in the stored image (normally referred to as (1,1)) came from the position (1,1024) in the original image. These original values are used for the following IMAGE_MAP_PROJECTION keywords in the PDS label for the stored image:

sample_first_pixel = 1
sample_last_pixel = 1024
line_first_pixel = 1024
line_last_pixel = 1

Now, given a pixel on the original image, P(X,Y) = (2,2) determine its location (P') in the stored image.

sample offset = (8 * (2 - 1) * 1024) / (8 * (1024 - 1 + 1)) = 1
line offset = ((2 - 1024) * 1024) / (1 - 1024 + 1) = (-1022)

Therefore, P' is located at (2, 1023) which is 1 byte from the first sample, and 1022 bytes (in the negative direction) from the first line in the stored image. See diagram above.

Required Keywords

1. MAP_PROJECTION_TYPE
2. A_AXIS_RADIUS
3. B_AXIS_RADIUS
4. C_AXIS_RADIUS
5. FIRST_STANDARD_PARALLEL
6. SECOND_STANDARD_PARALLEL
7. POSITIVE_LONGITUDE_DIRECTION
8. CENTER_LATITUDE
9. CENTER_LONGITUDE
10. REFERENCE_LATITUDE
11. REFERENCE_LONGITUDE
12. LINE_FIRST_PIXEL
13. LINE_LAST_PIXEL
14. SAMPLE_FIRST_PIXEL
15. SAMPLE_LAST_PIXEL
16. MAP_PROJECTION_ROTATION
17. MAP_RESOLUTION
18. MAP_SCALE
19. MAXIMUM_LATITUDE
20. MINIMUM_LATITUDE
21. EASTERNMOST_LONGITUDE
22. WESTERNMOST_LONGITUDE
23. LINE_PROJECTION_OFFSET
24. SAMPLE_PROJECTION_OFFSET
25. COORDINATE_SYSTEM_TYPE
26. COORDINATE_SYSTEM_NAME

Optional Keywords

1. DATA_SET_ID
2. IMAGE_ID
3. HORIZONTAL_FRAMELET_OFFSET
4. VERTICAL_FRAMELET_OFFSET

Required Objects

1. DATA_SET_MAP_PROJECTION

Optional Objects

None

Example

```
PDS_VERSION_ID                         = PDS3

/* File characteristics */
RECORD_TYPE                            = STREAM

/* Identification data elements */
DATA_SET_ID                             = "MGN-V-RDRS-5-GVDR-V1.0"
DATA_SET_NAME                          = "MAGELLAN VENUS RADAR SYSTEM GLOBAL DATA RECORD
V1.0"
PRODUCT_ID                             = "IMP-NORTH.100"

MISSION_NAME                           = "MAGELLAN"
SPACECRAFT_NAME                        = "MAGELLAN"
INSTRUMENT_NAME                        = "RADAR SYSTEM"
TARGET_NAME                            = "VENUS"

ORBIT_START_NUMBER                     = 376
ORBIT_STOP_NUMBER                      = 4367
START_TIME                             = "N/A"
STOP_TIME                              = "N/A"
SPACECRAFT_CLOCK_START_COUNT           = "N/A"
SPACECRAFT_CLOCK_STOP_COUNT            = "N/A"

PRODUCT_CREATION_TIME                   = 1994-05-07T22:09:27.000
PRODUCT_RELEASE_DATE                   = 1994-05-13
PRODUCT_SEQUENCE_NUMBER                = 00000
PRODUCT_VERSION_TYPE                   = "PRELIMINARY"

SOURCE_DATA_SET_ID                      = {"MGN-V-RDRS-5-SCVDR-V1.0",
"MGN-V-RDRS-CDR-ALT/RAD-V1.0"}
SOURCE_PRODUCT_ID                      = {"SCVDR.00376-00399.1","SCVDR.00400-00499.1",
"SCVDR.01100-01199.1","SCVDR.01200-01299.1","SCVDR.01300-01399.1",
"SCVDR.01400-01499.1","SCVDR.01500-01599.1","SCVDR.01600-01699.1",
"SCVDR.01700-01799.1","SCVDR.01800-01899.1","SCVDR.01900-01999.1",
"ARCDRCD.001;2","ARCDRCD.002;1","ARCDRCD.003;1","ARCDRCD.004;1",
"ARCDRCD.005;1","ARCDRCD.006;1","ARCDRCD.007;1","ARCDRCD.008;1",
"ARCDRCD.017;1","ARCDRCD.018;1","ARCDRCD.019;1"}

SOFTWARE_FLAG                          = "Y"

PRODUCER_FULL_NAME                     = "MICHAEL J. MAURER"
PRODUCER_INSTITUTION_NAME              = "STANFORD CENTER FOR RADAR ASTRONOMY"
PRODUCER_ID                            = "SCRA"
DESCRIPTION                             = "This file contains a single
  IMAGE_MAP_PROJECTION data object with an attached PDS label."

/* Data object definitions */
OBJECT                                 = IMAGE_MAP_PROJECTION
  ^DATA_SET_MAP_PROJECTION             = "DSMAP.CAT"
  COORDINATE_SYSTEM_TYPE               = "BODY-FIXED ROTATING"
  COORDINATE_SYSTEM_NAME               = "PLANETOCENTRIC"
  MAP_PROJECTION_TYPE                 = "STEREOGRAPHIC"
  A_AXIS_RADIUS                       = 6051.0 <KM>
  B_AXIS_RADIUS                       = 6051.0 <KM>
  C_AXIS_RADIUS                       = 6051.0 <KM>
  FIRST_STANDARD_PARALLEL             = "N/A"
  SECOND_STANDARD_PARALLEL            = "N/A"
  POSITIVE_LONGITUDE_DIRECTION        = "EAST"
  CENTER_LATITUDE                     = 90
  CENTER_LONGITUDE                    = 0
  REFERENCE_LATITUDE                  = "N/A"
  REFERENCE_LONGITUDE                 = "N/A"
  LINE_FIRST_PIXEL                    = 1
  LINE_LAST_PIXEL                     = 357
  SAMPLE_FIRST_PIXEL                  = 1
  SAMPLE_LAST_PIXEL                   = 357
  MAP_PROJECTION_ROTATION             = 0
  MAP_RESOLUTION                      = 5.79478 <PIXEL/DEGREE>
  MAP_SCALE                           = 18.225 <KM/PIXEL>
  MAXIMUM_LATITUDE                    = 90.00
  MINIMUM_LATITUDE                    = 60.00
  EASTERNMOST_LONGITUDE               = 360.00
  WESTERNMOST_LONGITUDE               = 0.00
  LINE_PROJECTION_OFFSET              = 178
  SAMPLE_PROJECTION_OFFSET            = 178
END_OBJECT                            = IMAGE_MAP_PROJECTION
END
```

A.21 INDEX_TABLE

The INDEX_TABLE object is a specific type of a TABLE object that provides information about the data stored on an archive volume. The INDEX_TABLE contains one row for each data file (or data product label file, in the case where detached labels are used) on the volume. The table is formatted so that it may be read directly by many data management systems on various host computers. All fields (columns) are separated by commas, and character fields are enclosed by double quotation marks. Each record ends in a carriage return/line feed sequence. This allows the table to be treated as a fixed length record file on hosts that support this file type and as a normal text file on other hosts.

There are two categories of columns for an Index table, identification and search. PDS data element names should be used as column names wherever appropriate.

The required columns are used for identification. The optional columns are data dependent and are used for search. For example, the following may be useful for searching:

Location (e.g. LATITUDE, LONGITUDE, ORBIT_NUMBER)
Time (e.g. START_TIME, SPACECRAFT_CLOCK_START_COUNT)
Feature (e.g. FEATURE_TYPE)
Observational characteristics (e.g. INCIDENCE_ANGLE)
Instrument characteristics (e.g. FILTER_NAME)

For archive volumes created before version 3.2 of the PDS standards, if the keyword INDEX_TYPE is not present, the value is defaulted to SINGLE, unless the Index's filename is given as CUMINDEX.TAB or axxCMIDX.TAB (with axx representing up to three alphanumeric characters).

If the keyword INDEXED_FILE_NAME is not present for a SINGLE index, the value is defaulted to "*.*" if attached labels are used, or "*.LBL" if detached labels are used. This indicates that the index encompasses all data product files on the volume.

If the INDEXED_FILE_NAME keyword is not present for a cumulative index, the default value is "*.TAB" for files in the INDEX subdirectory.

Note: See section 17.2 for information about the use of N/A, UNK and NULL in an INDEX table.

Required Keywords

1. INTERCHANGE_FORMAT
2. ROWS
3. COLUMNS
4. ROW_BYTES
5. INDEX_TYPE

Optional Keywords

1. NAME
2. DESCRIPTION
3. INDEXED_FILE_NAME
4. UNKNOWN_CONSTANT
5. NOT_APPLICABLE_CONSTANT

Required Objects

1. COLUMN

Optional Objects

None

Required COLUMN Objects (NAME=)

FILE_SPECIFICATION_NAME or PATH_NAME and FILE_NAME
PRODUCT_ID (**)
VOLUME_ID (*)
DATA_SET_ID (*)
PRODUCT_CREATION_TIME (*)
LOGICAL_VOLUME_PATH_NAME (must be used with PATH_NAME
and FILE_NAME for a logical volume) (*)

(*) If the value is constant across the data in the index table, this keyword can appear in the index table's label.
If the value is not constant, then a column of the given name must be used.

(**) PRODUCT_ID is not required if it has the same value as FILE_NAME
or FILE_SPECIFICATION_NAME.

Required Keywords (for Required COLUMN Objects)

NAME
DATA_TYPE
START_BYTE
BYTES
DESCRIPTION

Optional COLUMN Objects (NAME=)

MISSION_NAME
INSTRUMENT_NAME (or ID)
INSTRUMENT_HOST_NAME (or ID) (or SPACECRAFT_NAME or ID)
TARGET_NAME
PRODUCT_TYPE
MISSION_PHASE_NAME
VOLUME_SET_ID
START_TIME
STOP_TIME
SPACECRAFT_CLOCK_START_COUNT
SPACECRAFT_CLOCK_STOP_COUNT
any other search columns

Example (see additional example in A.27.1)

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                        = PDS3

RECORD_TYPE                            = FIXED_LENGTH
RECORD_BYTES                           = 180
FILE_RECORDS                           = 220
DESCRIPTION                            = "INDEX.TAB lists all data files on this volume"
^INDEX_TABLE                           = "INDEX.TAB"

OBJECT                                = INDEX_TABLE
  INTERCHANGE_FORMAT                  = ASCII
  ROW_BYTES                           = 180
  ROWS                                = 220
  COLUMNS                             = 9
  INDEX_TYPE                          = SINGLE
  INDEXED_FILE_NAME                   = {"*.AMD","*.ION","*.TIM","*.TRO",
                                      "*.WEA","*.LIT","*.MIF","*.MPD",
                                       "*.ODF","*.ODR","*.ODS","*.SFO",
                                       "*.SOE","*.TDF"}

  OBJECT                              = COLUMN
    NAME                              = VOLUME_ID
    DESCRIPTION                       = "Identifies the volume containing  the named file"
    DATA_TYPE                         = CHARACTER
    START_BYTE                        = 2
    BYTES                             = 9
  END_OBJECT                          = COLUMN

  OBJECT                              = COLUMN
    NAME                              = DATA_SET_ID
    DESCRIPTION                       = "The data set identifier. Acceptable values include
                                       ’MO-M-RSS-1-OIDR-V1.0’"
    DATA_TYPE                         = CHARACTER
    START_BYTE                        = 14
    BYTES                             = 25
  END_OBJECT                          = COLUMN

  OBJECT                              = COLUMN
    NAME                               = PATH_NAME
    DESCRIPTION                       = "Path to directory containing file.
                                      Acceptable values include:
                                       'AMD ',
                                       'ION ',
                                       'TIM ',
                                       'TRO',
                                       'WEA ',
                                       'LIT  ',
                                       'MIF ',
                                       'MPD ',
                                       'ODF ',
                                      'ODR ',
                                      'ODS ',
                                       'SFO ',
                                       'SOE', and
                                       'TDF '."
    DATA_TYPE                       = CHARACTER
    START_BYTE                      = 42
    BYTES                            = 9
  END_OBJECT                         = COLUMN

  OBJECT                            = COLUMN
    NAME                            = FILE_NAME
    DESCRIPTION                     = "Name of file in archive"
    DATA_TYPE                       = CHARACTER
    START_BYTE                      = 54
    BYTES                           = 12
  END_OBJECT                        = COLUMN

  OBJECT                            = COLUMN
    NAME                            = PRODUCT_ID
    DESCRIPTION                     = "Original file name on MO PDB or  SOPC"
    DATA_TYPE                       = CHARACTER
    START_BYTE                      = 69
    BYTES                           = 33
  END_OBJECT                        = COLUMN

  OBJECT                            = COLUMN
    NAME                            = START_TIME
    DESCRIPTION                     = "Time at which data in the file begin given in the format
                                    'YYYY-MM-DDThh:mm:ss'."
    DATA_TYPE                       = CHARACTER
    START_BYTE                      = 105
    BYTES                           = 19
  END_OBJECT                        = COLUMN

  OBJECT                            = COLUMN
    NAME                            = STOP_TIME
    DESCRIPTION                     = "Time at which data in the file end  given in the format
                                     'YYYY-MM-DDThh:mm:ss'."
    DATA_TYPE                       = CHARACTER
    START_BYTE                      = 127
    BYTES                           = 19
  END_OBJECT                        = COLUMN

  OBJECT                            = COLUMN
    NAME                            = PRODUCT_CREATION_TIME
    DESCRIPTION                     = "Date and time that file was created."
    DATA_TYPE                       = CHARACTER
    START_BYTE                      = 149
    BYTES                           = 19
  END_OBJECT                        = COLUMN

  OBJECT                            = COLUMN
    NAME                            = FILE_SIZE
    DESCRIPTION                     = "Number of bytes in file, not  including label."
    DATA_TYPE                       = "ASCII INTEGER"
    START_BYTE                      = 170
    BYTES                           = 9
  END_OBJECT                        = COLUMN

END_OBJECT                          = INDEX_TABLE
END
```

A.22 PALETTE

The PALETTE object, a sub-class of the table object, contains entries which represent color assignments for SAMPLE values contained in an IMAGE.

If the palette is stored in an external file from the data file, then the palette should be stored in ASCII format as 256 ROWS, each composed of 4 COLUMNS. The first column contains the SAMPLE value (0 to 255 for an 8-bit SAMPLE), and the remaining 3 COLUMNS contains the relative amount (a value from 0 to 255) of each primary color to be assigned for that SAMPLE value.

If the palette is stored in the data file, then it should be stored in BINARY format as 256 consecutive 8-bit values for each primary color (RED, GREEN, BLUE) resulting in a 768 byte record.

Required Keywords

1. INTERCHANGE_FORMAT
2. ROWS
3. ROW_BYTES
4. COLUMNS

Optional Keywords

1. DESCRIPTION
2. NAME

Required Objects

1. COLUMN

Optional Objects

None

Example

The examples below depict the differences between the two types of PALETTE objects. The first is an example of an ASCII PALETTE object, and the second is an example of the BINARY PALETTE object.

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                   = PDS3
RECORD_TYPE                      = FIXED_LENGTH
RECORD_BYTES                     = 80
FILE_RECORDS                     = 256
```

```
^PALETTE                          = "PALETTE.TAB"
 /* Image Palette description  */
SPACECRAFT_NAME                   = MAGELLAN
MISSION_PHASE_NAME                = PRIMARY_MISSION
TARGET_NAME                       = VENUS
PRODUCT_ID                        ="GEDR-MERC.1;2"
IMAGE_ID                          ="GEDR-MERC.1;2"
INSTRUMENT_NAME                   ="RADAR SYSTEM"
PRODUCT_CREATION_TIME             = 1995-01-01T00:00:00
NOTE                              = "Palette for browse image"

/* Description of an ASCII PALETTE object */

OBJECT                            = PALETTE
INTERCHANGE_FORMAT                = ASCII
ROWS                              = 256
ROW_BYTES                         = 80
COLUMNS                           = 4
OBJECT                            = COLUMN
NAME                              = SAMPLE
DESCRIPTION                       ="DN value for red, green, blue intensities"
DATA_TYPE                         = INTEGER
START_BYTE                        = 1
BYTES                             = 3
END_OBJECT
OBJECT                            = COLUMN
NAME                              = RED
DESCRIPTION                       = "Red intensity (0 - 255)"
DATA_TYPE                         = INTEGER
START_BYTE                        = 6
BYTES                             = 3
END_OBJECT
OBJECT                            = COLUMN
NAME                              = GREEN
DESCRIPTION                       = "Green intensity (0 - 255)"
DATA_TYPE                         = INTEGER
START_BYTE                        = 11
BYTES                             = 3
END_OBJECT
OBJECT                            = COLUMN
NAME                              = BLUE
DESCRIPTION                       = "Blue intensity (0 - 255)"
DATA_TYPE                         = INTEGER
START_BYTE                        = 16
BYTES                             = 3
END_OBJECT
END_OBJECT
END

------------------------------------------------------------

/* Description of a BINARY PALETTE object */

OBJECT                            = PALETTE
INTERCHANGE_FORMAT                = BINARY
ROWS                              = 1
ROW_BYTES                         = 768
COLUMNS                           = 3
```

```
OBJECT                          = COLUMN
NAME                            = RED
DATA_TYPE                       = UNSIGNED_INTEGER
START_BYTE                      = 1
ITEMS                           = 256
ITEM_BYTES                      = 1
END_OBJECT                      = COLUMN

OBJECT                          = COLUMN
NAME                            = GREEN
DATA_TYPE                       = UNSIGNED_INTEGER
START_BYTE                      = 257
ITEMS                           = 256
ITEM_BYTES                      = 1
END_OBJECT                      = COLUMN

OBJECT                          = COLUMN
NAME                            = BLUE
DATA_TYPE                       = UNSIGNED_INTEGER
START_BYTE                      = 513
ITEMS                           = 256
ITEM_BYTES                      = 1
END_OBJECT                      = COLUMN
END_OBJECT                      = PALETTE
END
```

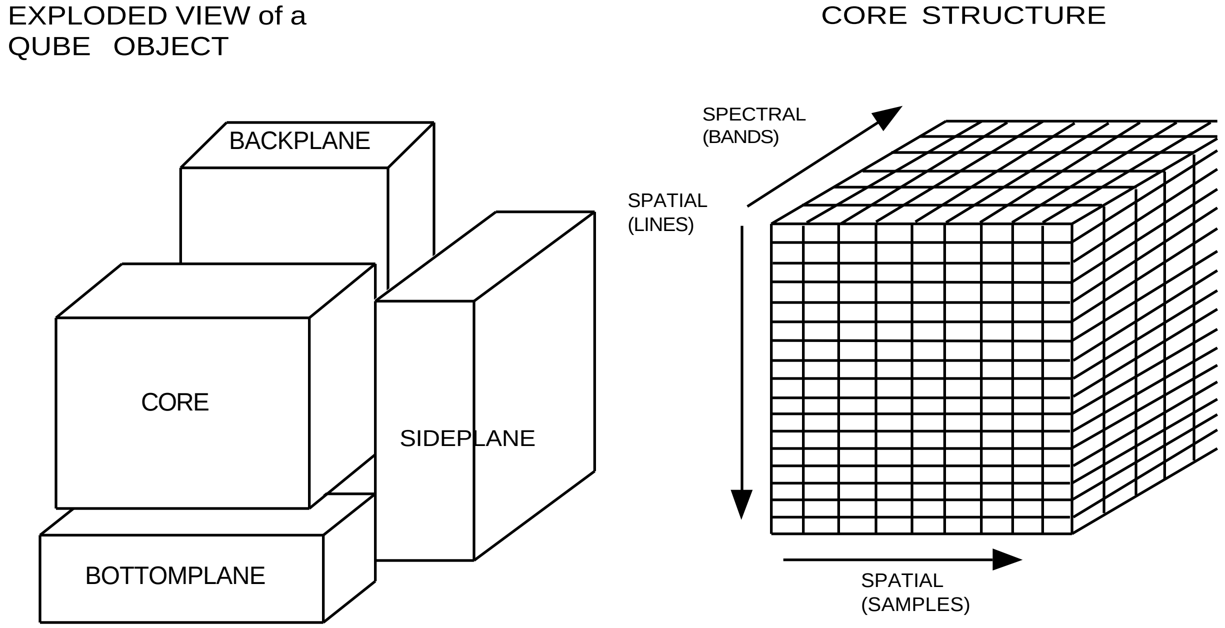
A.23 QUBE

A generalized QUBE object is a multidimensional array (called the core) of sample values in multiple dimensions. The core is homogeneous, and consists of unsigned byte, signed halfword or floating point fullword elements. QUBEs of one to three dimensions may have optional suffix areas in each axis. The suffix areas may be heterogeneous, with elements of different types, but each suffix pixel is always allocated a fullword. Special values may be defined for the core and the suffix areas to designate missing values and several kinds of invalid values, such as instrument and representation saturation.

The QUBE is the principal data structure of the ISIS (Integrated Software for Imaging Spectrometers) system. A frequently used specialization of the QUBE object is the ISIS Standard Qube, which is a three-dimensional QUBE with two spatial dimensions and one spectral dimension. Its axes have the interpretations 'sample', 'line' and 'band'. Three physical storage orders are allowed: band-sequential, line_interleaved (band-interleaved-by-line) and sample_interleaved (band-interleaved-by-pixel).

An example of a Standard ISIS Qube is a spectral image qube containing data from an imaging spectrometer. Such a qube is simultaneously a set of images (at different wavelengths) of the same target area, and a set of spectra at each point of the target area. Typically, suffix areas in such a qube are confined to 'backplanes' containing geometric or quality information about individual spectra, i.e. about the set of corresponding values at the same pixel location in each band.

The following diagram illustrates the general structure of a Standard ISIS Qube. Note that this is a conceptual or “logical” view of the qube.


EXPLODED VIEW of a QUBE OBJECT

CORE STRUCTURE

Figure A.3: Exploded View of a Qube Object

Some special requirements are imposed by the ISIS system. A QUBE object must be associated with a HISTORY object. (Other objects, such as HISTOGRAMs, IMAGEs, PALETTEs and TABLEs which contain statistics, display parameters, engineering values or other ancillary data, are optional.) A special element, FILE_STATE, is required in the implicit FILE object. Some label information is organized into GROUPs, such as BAND_BIN and IMAGE_MAP_PROJECTION. The BAND_BIN group contains essential wavelength information, and is required for Standard ISIS Qubes.

The ISIS system includes routines for reading and writing files containing QUBE objects. Both 'logical' access, independent of actual storage order, and direct 'physical' access are provided for Standard ISIS Qubes. Only physical access is provided for generalized QUBEs. Most ISIS application programs operate on Standard ISIS Qubes. Arbitrary subqubes ('virtual' qubes) of existing qubes may be specified for most of these programs. In addition, ISIS includes software for handling Tables (an ISIS variant of the PDS Table object) and Instrument Spectral Libraries.

For a complete description, refer to the most recent version of 'ISD: ISIS System Design, Build 2', obtainable from the PDS Operator.

NOTE: The following required and optional elements of the QUBE object are ISIS-specific. Since the ISIS system was designed before the current version of the Planetary Science Data Dictionary, some of the element names conflict with current PDS nomenclature standards.

Required Keywords (Generalized Qube and Standard ISIS Qube)

AXES	Number of axes or dimensions of qube [integer]
AXIS_NAME	Names of axes [sequence of 1-6 literals] (BAND, LINE, SAMPLE) for Standard Qube
CORE_ITEMS	Core dimensions of axes [seq of 1-6 integers]
CORE_ITEM_BYTES	Core element size [integer bytes: {1, 2, 4}]
CORE_ITEM_TYPE	Core element type [literal: {UNSIGNED_INTEGER, INTEGER, REAL}]
CORE_BASE	Base value of core item scaling [real]
CORE_MULTIPLIER	Multiplier for core item scaling [real] 'true' value = base + multiplier * 'stored' value (base = 0.0 and multiplier = 1.0 for REALs)
SUFFIX_BYTES	Storage allocation of suffix elements [integer: always 4]
SUFFIX_ITEMS	Suffix dimensions of axes [seq of 1-6 integers]
CORE_VALID_MINIMUM	Minimum valid core value -- values below this value are reserved for 'special' values, of which 5 are currently assigned [integer or non-decimal integer: these values are fixed by ISIS convention for each allowable item type and size -- see ISD for

	details]
CORE_NULL	Special value indicating 'invalid' data
CORE_LOW_INSTR_SATURATION	Special value indicating instrument saturation at the low end
CORE_HIGH_INSTR_SATURATION	Special value indicating instrument saturation at the high end
CORE_LOW_REPR_SATURATION	Special value indicating representation saturation at the low end
CORE_HIGH_REPR_SATURATION	Special value indicating representation saturation at the high end

Required Keywords (Standard ISIS Qube) and Optional Keywords (Generalized Qube)

CORE_NAME	Name of value stored in core of qube [literal, e.g. SPECTRAL_RADIANCE]
CORE_UNIT	Unit of value stored in core of qube [literal]
BAND_BIN_CENTER	Wavelengths of bands in a Standard Qube [sequence of reals]
BAND_BIN_UNIT	Unit of wavelength [literal, e.g. MICROMETER]
BAND_BIN_ORIGINAL_BAND	Original band numbers, referring to a Qube of which the current qube is a subqube. In the original qube, these are sequential integers.[sequence of integers]

Optional Keywords (Generalized Qube and Standard ISIS Qube)

BAND_BIN_WIDTH	Width (at half height) of spectral response of bands [sequence of reals]
BAND_BIN_STANDARD_DEVIATION	Standard deviation of spectrometer values at each band [sequence of reals]
BAND_BIN_DETECTOR	Instrument detector number of band, where relevant [sequence of integers]
BAND_BIN_GRATING_POSITION	Instrument grating position of band, where relevant [sequence of integers]

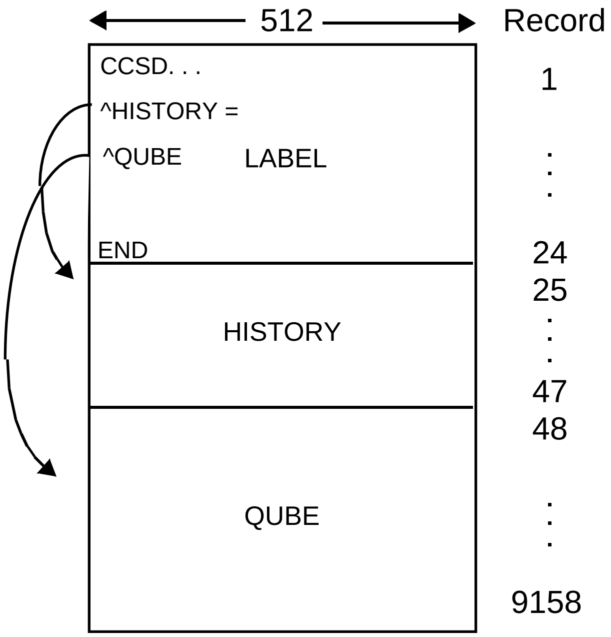
Required Keywords (for each suffix present in a 1-3 dimensional qube).
Note: These must be prefixed by the specific AXIS_NAME. These are SAMPLE, LINE and BAND for Standard ISIS Qubes. Only the commonly used BAND variants are shown:

BAND_SUFFIX_NAME	Names of suffix items [sequence of literals]
BAND_SUFFIX_UNIT	Units of suffix items [sequence of literals]
BAND_SUFFIX_ITEM_BYTES	Suffix item sizes [sequence of integer bytes {1, 2, 4}]
BAND_SUFFIX_ITEM_TYPE	Suffix item types [sequence of literals:

	{UNSIGNED_INTEGER, INTEGER, REAL, ...}]
BAND_SUFFIX_BASE	Base values of suffix item scaling [sequence of reals] (see corresponding core element)
BAND_SUFFIX_MULTIPLIER	Multipliers for suffix item scaling [sequence of reals] (see corresponding core element)
BAND_SUFFIX_VALID_MINIMUM	Minimum valid suffix values
BAND_SUFFIX_NULL	...and assigned special values
BAND_SUFFIX_LOW_INSTR_SAT	[sequences of integers or reals]
BAND_SUFFIX_HIGH_INSTR_SAT	(see corresponding core
BAND_SUFFIX_LOW_REPR_SAT	element definitions for
BAND_SUFFIX_HIGH_REPR_SAT	details)

Example

The following label describes ISIS qube data from the Galileo NIMS experiment. The qube contains 17 bands of NIMS fixed-map mode raw data numbers and 9 backplanes of ancillary information. In other modes, NIMS can produce data qubes of 34, 102, 204 and 408 bands.



```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID = PDS3
/* File Structure */
```

```
RECORD_TYPE                         = FIXED_LENGTH
RECORD_BYTES                        = 512
FILE_RECORDS                        =   9158
LABEL_RECORDS                       =     24
FILE_STATE                          = CLEAN

^HISTORY                            =     25
OBJECT                              = HISTORY
END_OBJECT                          = HISTORY

^QUBE                               =    48
OBJECT                              = QUBE

/*  Qube structure: Standard ISIS Cube of NIMS Data */

 AXES                               = 3
 AXIS_NAME                          = (SAMPLE,LINE,BAND)

/*  Core description */

 CORE_ITEMS                         = (229,291,17)
 CORE_ITEM_BYTES                    = 2
 CORE_ITEM_TYPE                     = VAX_INTEGER
 CORE_BASE                          = 0.0
 CORE_MULTIPLIER                    = 1.0
 CORE_VALID_MINIMUM                 = -32752
 CORE_NULL                          = -32768
 CORE_LOW_REPR_SATURATION           = -32767
 CORE_LOW_INSTR_SATURATION          = -32766
 CORE_HIGH_INSTR_SATURATION         = -32765
 CORE_HIGH_REPR_SATURATION          = -32764
 CORE_NAME                          = RAW_DATA_NUMBER
 CORE_UNIT                          = DIMENSIONLESS

 PHOTOMETRIC_CORRECTION_TYPE        = NONE

/*  Suffix description  */

 SUFFIX_BYTES                       = 4
 SUFFIX_ITEMS                       = (0,0,9)

 BAND_SUFFIX_NAME                   = (LATITUDE,LONGITUDE,INCIDENCE_ANGLE,
   EMISSION_ANGLE,PHASE_ANGLE,SLANT_DISTANCE,INTERCEPT_ALTITUDE,
   PHASE_ANGLE_STD_DEV,RAW_DATA_NUMBER_STD_DEV)
 BAND_SUFFIX_UNIT                   = (DEGREE,DEGREE,DEGREE,DEGREE,DEGREE,KILOMETER,
   KILOMETER,DEGREE,DIMENSIONLESS)
 BAND_SUFFIX_ITEM_BYTES             = (4,4,4,4,4,4,4,4,4)
 BAND_SUFFIX_ITEM_TYPE              = (VAX_REAL,VAX_REAL,VAX_REAL,VAX_REAL,VAX_REAL,
     VAX_REAL,VAX_REAL,VAX_REAL,VAX_REAL)
 BAND_SUFFIX_BASE                   = (0.000000,0.000000,0.000000,0.000000,0.000000,
     0.000000,0.000000,0.000000,0.000000)
 BAND_SUFFIX_MULTIPLIER             = (1.000000,1.000000,1.000000,1.000000,1.000000,
     1.000000,1.000000,1.000000,1.000000)
 BAND_SUFFIX_VALID_MINIMUM          = (16#FFEFFFFF#,16#FFEFFFFF#,16#FFEFFFFF#,
     16#FFEFFFFF#,16#FFEFFFFF#,16#FFEFFFFF#,16#FFEFFFFF#,16#FFEFFFFF#,
     16#FFEFFFFF#)
 BAND_SUFFIX_NULL                   = (16#FFFFFFFF#,16#FFFFFFFF#,16#FFFFFFFF#,16#FFFFFFFF#,
     16#FFFFFFFF#,16#FFFFFFFF#,16#FFFFFFFF#,16#FFFFFFFF#,16#FFFFFFFF#)
 BAND_SUFFIX_LOW_REPR_SAT              = (16#FFFEFFFF#,16#FFFEFFFF#,16#FFFEFFFF#,
    16#FFFEFFFF#,16#FFFEFFFF#,16#FFFEFFFF#,16#FFFEFFFF#,16#FFFEFFFF#,
    16#FFFEFFFF#)
 BAND_SUFFIX_LOW_INSTR_SAT             = (16#FFFDFFFF#,16#FFFDFFFF#,16#FFFDFFFF#,
    16#FFFDFFFF#,16#FFFDFFFF#,16#FFFDFFFF#,16#FFFDFFFF#,16#FFFDFFFF#,
    16#FFFDFFFF#)
 BAND_SUFFIX_HIGH_INSTR_SAT            = (16#FFFCFFFF#,16#FFFCFFFF#,16#FFFCFFFF#,
    16#FFFCFFFF#,16#FFFCFFFF#,16#FFFCFFFF#,16#FFFCFFFF#,16#FFFCFFFF#,
    16#FFFCFFFF#)
 BAND_SUFFIX_HIGH_REPR_SAT             = (16#FFFBFFFF#,16#FFFBFFFF#,16#FFFBFFFF#,
    16#FFFBFFFF#,16#FFFBFFFF#,16#FFFBFFFF#,16#FFFBFFFF#,16#FFFBFFFF#,
    16#FFFBFFFF#)
 BAND_SUFFIX_NOTE                      = "
The backplanes contain 7 geometric parameters, the standard deviation   of one of them, the standard deviation of a selected data
band, and   0 to 10 'spectral index' bands, each a user-specified function of the   data bands.  (See the BAND_SUFFIX_NAME
values.)

  Longitude ranges from 0 to 360 degrees, with positive direction specified by POSITIVE_LONGITUDE_DIRECTION in the
IMAGE_MAP_PROJECTION group.

  INTERCEPT_ALTITUDE contains values for the DIFFERENCE between   the length of the normal from the center of the target
body to the line of sight AND the radius of the target body.  On-target points   have zero values.  Points beyond the maximum
expanded radius have   null values.  This plane thus also serves as a set of 'off-limb'   flags.  It is meaningful only for the
ORTHOGRAPHIC and   POINT_PERSPECTIVE projections; otherwise all values are zero.    The geometric standard deviation
backplane contains the standard   deviation of the geometry backplane indicated in its NAME, except   that the special value
16#FFF9FFFF# replaces the standard deviation   where the corresponding core pixels have been 'filled'.

The data band standard deviation plane is computed for the NIMS data   band specified by
STD_DEV_SELECTED_BAND_NUMBER.  This may be either   a raw data number, or spectral radiance, whichever is indicated
by   CORE_NAME.

The (optional) spectral index bands were generated by the Vicar F2   program.  The corresponding BAND_SUFFIX_NAME is an
abbreviated formula for the function used, where Bn should be read 'NIMS data band n'.  For example: B4/B8 represents the ratio
of bands 4 and 8."

STD_DEV_SELECTED_BAND_NUMBER       = 9

/*  Data description: general */

DATA_SET_ID                           = "GO-V-NIMS-4-MOSAIC-V1.0"
PRODUCT_ID                            = "XYZ"
SPACECRAFT_NAME                       = GALILEO_ORBITER
MISSION_PHASE_NAME                    = VENUS_ENCOUNTER
INSTRUMENT_NAME                       = NEAR_INFRARED_MAPPING_SPECTROMETER
INSTRUMENT_ID                         = NIMS
^INSTRUMENT_DESCRIPTION               = "NIMSINST.TXT"

TARGET_NAME                           = VENUS
START_TIME                            = 1990-02-10T01:49:58Z
STOP_TIME                             = 1990-02-10T02:31:52Z
NATIVE_START_TIME                     =   180425.85
NATIVE_STOP_TIME                      =   180467.34
OBSERVATION_NAME                      = 'VPDIN1'
OBSERVATION_NOTE                      = "VPDIN1 / Footprint, Limbfit, Height=50"

INCIDENCE_ANGLE                       = 160.48
EMISSION_ANGLE                        = 14.01
PHASE_ANGLE                           = 147.39
SUB_SOLAR_AZIMUTH                     = -174.74
```

```
SUB_SPACECRAFT_AZIMUTH               = -0.80
MINIMUM_SLANT_DISTANCE               =  85684.10
MAXIMUM_SLANT_DISTANCE               = 103175.00
MIN_SPACECRAFT_SOLAR_DISTANCE        = 1.076102e+08
MAX_SPACECRAFT_SOLAR_DISTANCE        = 1.076250e+08

/*  Data description: instrument status  */

INSTRUMENT_MODE_ID                   = FIXED_MAP
GAIN_MODE_ID                         = 2
CHOPPER_MODE_ID                      = REFERENCE
START_GRATING_POSITION               = 16
OFFSET_GRATING_POSITION              = 04

MEAN_FOCAL_PLANE_TEMPERATURE         = 85.569702
MEAN_RAD_SHIELD_TEMPERATURE          = 123.636002
MEAN_TELESCOPE_TEMPERATURE           = 139.604996
MEAN_GRATING_TEMPERATURE             = 142.580002
MEAN_CHOPPER_TEMPERATURE             = 142.449997
MEAN_ELECTRONICS_TEMPERATURE         = 287.049988

GROUP                                = BAND_BIN

/*  Spectral axis description */

BAND_BIN_CENTER                      = (0.798777,0.937873,1.179840,1.458040,1.736630,
   2.017250,2.298800,2.579060,2.864540,3.144230,3.427810,3.710640,
   3.993880,4.277290,4.561400,4.843560,5.126080)
BAND_BIN_UNIT                        = MICROMETER
BAND_BIN_ORIGINAL_BAND               = (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,
17)
BAND_BIN_GRATING_POSITION            = (16,16,16,16,16,16,16,16,16,16,16,16,
16,16,16,16,16)
BAND_BIN_DETECTOR                    = (1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17)
END_GROUP                            = BAND_BIN

GROUP                                = IMAGE_MAP_PROJECTION
/* Projection description */
MAP_PROJECTION_TYPE                  = OBLIQUE_ORTHOGRAPHIC
MAP_SCALE                            = 45.000
MAP_RESOLUTION                       =  2.366
CENTER_LATITUDE                      =  12.00
CENTER_LONGITUDE                     = 350.00
LINE_PROJECTION_OFFSET               = 149.10
SAMPLE_PROJECTION_OFFSET             =  85.10
MINIMUM_LATITUDE                     =  11.71
MAXIMUM_LATITUDE                     =  13.62
MINIMUM_LONGITUDE                    = 349.62
MAXIMUM_LONGITUDE                    = 351.72
POSITIVE_LONGITUDE_DIRECTION         = EAST
A_AXIS_RADIUS                        = 6101.000000
B_AXIS_RADIUS                        = 6101.000000
C_AXIS_RADIUS                        = 6101.000000
REFERENCE_LATITUDE                   = 0.000000
REFERENCE_LONGITUDE                  = 0.000000
MAP_PROJECTION_ROTATION              =   0.00
LINE_FIRST_PIXEL                     = 1
LINE_LAST_PIXEL                      = 229
SAMPLE_FIRST_PIXEL                   = 1
```

```
SAMPLE_LAST_PIXEL                  = 291
END_GROUP                          = IMAGE_MAP_PROJECTION

END_OBJECT                         = QUBE
END
```

A.24 SERIES

The SERIES object is a sub-class of the TABLE object. It is used for storing a sequence of measurements organized in a specific way (e.g. ascending time, radial distances). The current version uses the same physical format specification as the TABLE object, but includes sampling parameter information that describes the variation between elements in the series.

The sampling parameter keywords are required for the SERIES object and may be optional for one or more COLUMN sub-objects, depending on the data organization.

The sampling parameter keywords in the SERIES object represent the variation between the ROWS of data. For data that vary regularly between each row, the SAMPLING_PARAMETER_INTERVAL keyword defines this regularity. For data in which rows are irregularly spaced, the SAMPLING_PARAMETER_INTERVAL keyword is “N/A”, and the actual sampling parameter values are included in the data itself and identified as a column in the series. An example of this is a file of time series data with rows ordered by a time column (or set of columns).

For data that vary regularly between items of a single column, sampling parameter keywords appear as part of the COLUMN sub-object. Data sampled at irregular intervals described as separate columns may also provide sampling parameter information specific to each column.

Optional MINIMUM_SAMPLING_PARAMETER and MAXIMUM_SAMPLING_PARAMETER keywords should be added whenever possible to indicate the range in which the data was sampled. For data sampled at a single point rather than over a range, both the MINIMUM_SAMPLING_PARAMETER and MAXIMUM_SAMPLING_PARAMETER are set to the specific value. For TIME_SERIES data, where the sampling parameter specified is time, these keywords are not used.

Required Keywords

1. INTERCHANGE_FORMAT
2. ROWS
3. COLUMNS
4. ROW_BYTES
5. SAMPLING_PARAMETER_NAME
6. SAMPLING_PARAMETER_UNIT
7. SAMPLING_PARAMETER_INTERVAL

Optional Keywords

1. NAME
2. ROW_PREFIX_BYTES
3. ROW_SUFFIX_BYTES
4. MINIMUM_SAMPLING_PARAMETER
5. MAXIMUM_SAMPLING_PARAMETER

6. DERIVED_MINIMUM
7. DERIVED_MAXIMUM
8. DESCRIPTION

Required Objects

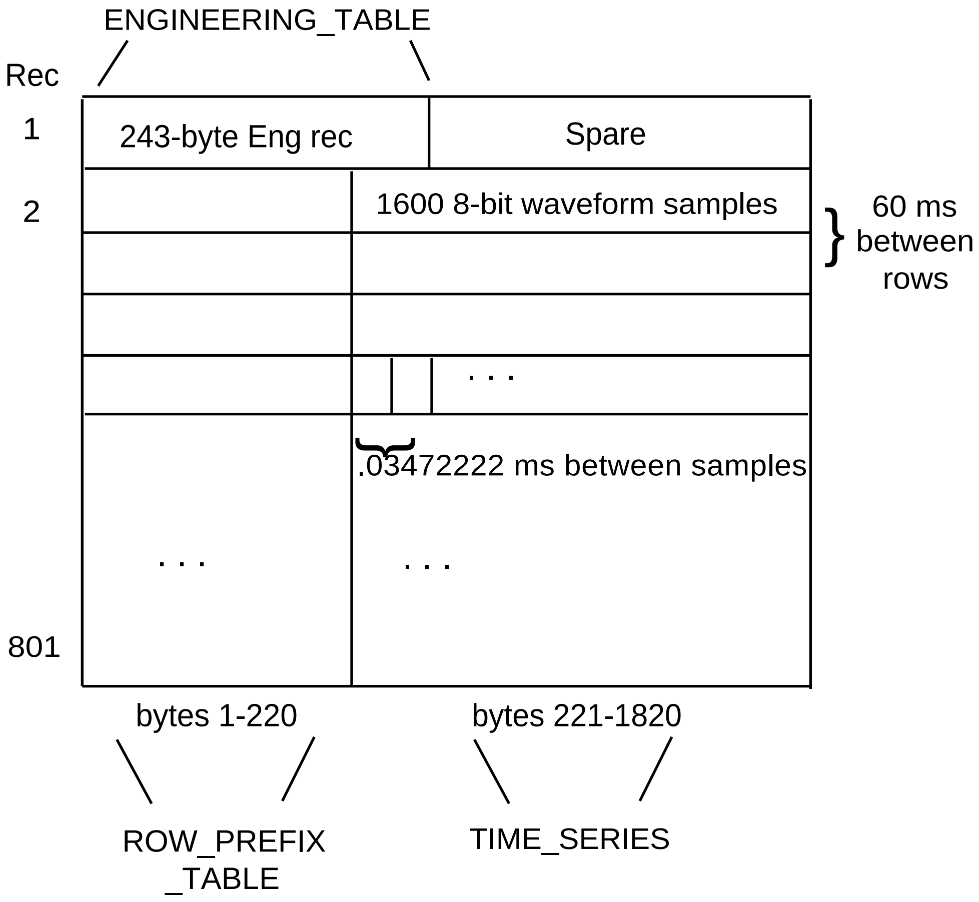
1. COLUMN

Optional Objects

1. CONTAINER

Example

This example illustrates the use of the SERIES object for data that vary regularly in two ways. Rows of data in the SERIES occur at 60 millisecond intervals while the COLUMN occurs at .03472222 millisecond intervals.



```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                      = PDS3
RECORD_TYPE                         = FIXED_LENGTH
RECORD_BYTES                        = 1820
FILE_RECORDS                        = 801
^ENGINEERING_TABLE                  = ("C0900313.DAT", 1)
^ROW_PREFIX_TABLE                   = ("C0900313.DAT", 2)
^TIME_SERIES                        = ("C0900313.DAT", 2)
/* Observation description */
DATA_SET_ID                         = "VG2-N-PWS-2-EDR-WFRM-60MS-V1.0"
PRODUCT_ID                          = "C0900313.DAT"
PRODUCT_CREATION_TIME               = "UNK"
SPACECRAFT_NAME                     = VOYAGER_2
SPACECRAFT_CLOCK_START_COUNT        = "09003.13.002"
SPACECRAFT_CLOCK_STOP_COUNT         = "09003.13.002"
EARTH_RECEIVED_TIME                 = 1989-159T13:35:00.121Z
START_TIME                          = 1989-157T14:16:56.979Z
STOP_TIME                           = "N/A"
MISSION_PHASE_NAME                  = NEPTUNE_ENCOUNTER
TARGET_NAME                         = NEPTUNE
/* Instrument description */
INSTRUMENT_NAME                     = PLASMA_WAVE_RECEIVER
INSTRUMENT_ID                       = PWS
SECTION_ID                          = WFRM
/* Object descriptions */
OBJECT                              = ENGINEERING_TABLE
INTERCHANGE_FORMAT                  = BINARY
ROWS                                = 1
COLUMNS                             = 106
ROW_BYTES                           = 243
ROW_SUFFIX_BYTES                    =1577
DESCRIPTION                         = "This table describes the format of the engineering record which is included as
the first record in  each PWS high rate waveform file.  This record contains the  first 242 bytes of data extracted from the Mission
and Test Imaging System (MTIS) header record on each file of an imaging EDR tape.  A 243rd byte containing some flag fields has
been added to the table for all data collected during the Neptune encounter."
  ^STRUCTURE                        = "ENGTAB.FMT"
END_OBJECT                          = ENGINEERING_TABLE

OBJECT                              = ROW_PREFIX_TABLE
INTERCHANGE_FORMAT                  = BINARY
ROWS                                = 800
COLUMNS                             = 47
ROW_BYTES                           = 220
ROW_SUFFIX_BYTES                    = 1600
DESCRIPTION                         = "This table describes the format of the engineering data associated with the
collection of each row of waveform data (1600 waveform samples)."
^STRUCTURE                          = "ROWPRX.FMT"
END_OBJECT                          = ROW_PREFIX_TABLE

OBJECT                              = TIME_SERIES
NAME                                = WAVEFORM_FRAME
INTERCHANGE_FORMAT                  = BINARY
ROWS                                = 799
COLUMNS                             = 1
ROW_BYTES                           = 1600
ROW_PREFIX_BYTES                    = 220
SAMPLING_PARAMETER_NAME             = TIME
```

```
SAMPLING_PARAMETER_UNIT              = SECOND
SAMPLING_PARAMETER_INTERVAL          = .06        /* 60 MS between rows */
DESCRIPTION                          = "This time_series consists of up to 800  records (or rows, lines) of PWS
waveform sample data.  Each record 2-801 of the file (or frame) contains 1600 waveform samples, prefaced by 220 bytes of MTIS
information.  The 1600 samples are collected in 55.56 msec followed by a 4.44 msec gap.  Each 60 msec interval constitutes a line
of waveform      samples.  Each file contains up to 800 lines of waveform samples for a 48 sec frame."

OBJECT                               = COLUMN
NAME                                 = WAVEFORM_SAMPLES
DATA_TYPE                            = MSB_UNSIGNED_INTEGER
START_BYTE                           = 221
BYTES                                = 1600
ITEMS                                = 1600
ITEM_BYTES                           = 1
SAMPLING_PARAMETER_NAME              = TIME
SAMPLING_PARAMETER_UNIT              = SECOND
SAMPLING_PARAMETER_INTERVAL          = 0.00003472222 /* time between samples */

OFFSET                               = -7.5
VALID_MINIMUM                        = 0
VALID_MAXIMUM                        = 15
DESCRIPTION                          = "The 1 byte waveform samples constitute an array of waveform measurements
which are encoded into binary values from 0 to 15 and may be re-mapped to reduce the artificial zero-frequency component. For
example,  stored values can be mapped to the following floating point  values.  The original 4-bit data samples have been repackaged
into 8-bit (1 byte) items without modification for archival purposes.\n

0 = -7.5             1 = -6.5             2 = -5.5             3 = -4.5
4 = -3.5             5 = -2.5             6 = -1.5             7 = -0.5
8 =0.5               9 =1.5               10=2.5               11 =3.5
12 =4.5              13 =5.5              14 =6.5              15 =7.5

    "
END_OBJECT                           = COLUMN
END_OBJECT                           = TIME_SERIES

END
```

A.25 SPECTRUM

The SPECTRUM object is a form of TABLE used for storing spectral measurements. The SPECTRUM object is assumed to have a number of measurements of the observation target taken in different SPECTRAL bands. The SPECTRUM object uses the same physical format specification as the TABLE object, but includes a SAMPLING PARAMETER definition which indicates the spectral region measured in successive COLUMNs or ROWs. The common sampling parameters for SPECTRUM objects are wavelength, frequency, or velocity.

A regularly sampled SPECTRUM can be stored either horizontally as a 1 row table with 1 column containing n samples (expressed as ITEMS=n), or vertically as a 1 column table with n rows where each ROW contains a sample of the spectrum. The vertical format allows additional columns to be defined for related parameters for each sample value (e.g. ERROR factors). These related columns can be described in a separate PREFIX or SUFFIX table.

An irregularly sampled SPECTRUM must be stored horizontally, with each specific spectral range identified as a separate column, and defined by a specific set of sampling parameter keywords for each column.

In the horizontal format, the sampling parameter specifications are included in the COLUMN definition. For a vertically defined SPECTRUM, the sampling parameter information is provided in the SPECTRUM object, since it is describing the spectral variation between the ROWs of the data.

Required Keywords

1. INTERCHANGE_FORMAT
2. ROWS
3. COLUMNS
4. ROW_BYTES

Optional Keywords

1. NAME
2. SAMPLING_PARAMETER_NAME
3. SAMPLING_PARAMETER_UNIT
4. SAMPLING_PARAMETER_INTERVAL
5. ROW_PREFIX_BYTES
6. ROW_SUFFIX_BYTES
7. MINIMUM_SAMPLING_PARAMETER
8. MAXIMUM_SAMPLING_PARAMETER
9. DERIVED_MINIMUM
10. DERIVED_MAXIMUM
11. DESCRIPTION

Required Objects

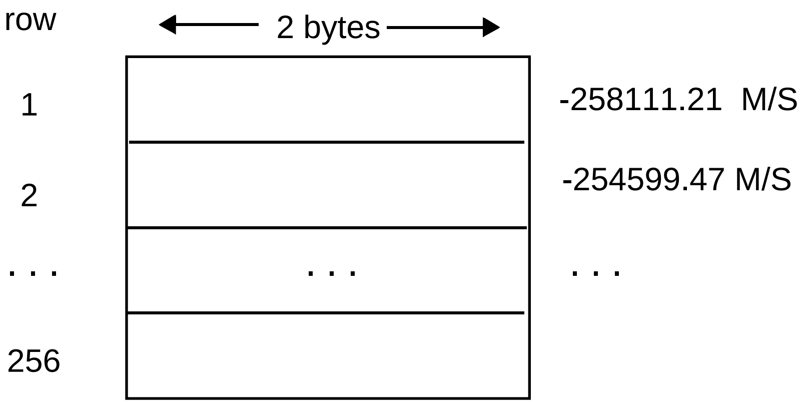
1. COLUMN

Optional Objects

1. CONTAINER

Example

This example illustrates a SPECTRUM data object stored in a vertical format. The data are regularly sampled at intervals of 99.09618 meters/second and data samples are stored in successive ROWS.



```
CCSD3ZF0000100000001NJPL3IFOPDSX00000001
PDS_VERSION_ID                    = PDS3
RECORD_TYPE                       = FIXED_LENGTH
RECORD_BYTES                      = 2
FILE_RECORDS                      = 256
PRODUCT_ID                        = "RSSL007.DAT"
DATA_SET_ID                       =  "IHW-C-RSSL-3-EDR-HALLEY-V1.0"
TARGET_NAME                       =  "HALLEY"
INSTRUMENT_HOST_NAME              = "IHW RADIO STUDIES NETWORK"
INSTRUMENT_NAME                   = "RADIO SPECTRAL LINE DATA"
OBSERVATION_ID                    = "621270"
START_TIME                        = 1985-11-10T00:43:12.000
STOP_TIME                         = 1985-11-10T00:43:12.000
PRODUCT_CREATION_TIME             = "UNK"
/* Record Pointer to Major Object */
^TOTAL_INTENSITY_SPECTRUM         = "RSSL0007.DAT"
/* Object Description */

OBJECT                            = SPECTRUM
INTERCHANGE_FORMAT                = BINARY
ROWS                              = 256
ROW_BYTES                         = 2
COLUMNS                           = 1
SAMPLING_PARAMETER_NAME           = "VELO_COM"
```

```
MINIMUM_SAMPLING_PARAMETER          = -1.268431E+04
SAMPLING_PARAMETER_INTERVAL         = 9.909618E+01
SAMPLING_PARAMETER_UNIT             = "METERS/SECOND"
DESCRIPTION                         = "Radio Studies; Spectral Line intensity spectrum.  Spectrum is organized as 1
column with 256 rows.  Each row contains a spectral value for the velocity derived from the sampling parameter information
associated with each row."

OBJECT                              = COLUMN
NAME                                = FLUX_DENSITY
DATA_TYPE                           = MSB_INTEGER
START_BYTE                          = 1
BYTES                               = 2
SCALING_FACTOR                      = 7.251200E-04
OFFSET                              = 0.000000E+01
DERIVED_MINIMUM                     = 2.380000E+01
DERIVED_MAXIMUM                     = 3.490000E+01
END_OBJECT                          = COLUMN
END_OBJECT                          = SPECTRUM

END
```

A.26 SPICE KERNEL

The SPICE_KERNEL object defines a single kernel (file) from a collection of SPICE Kernels. SPICE kernels provide ancillary data needed to support the planning and subsequent analysis of space science observations.

The SPICE system includes the software and documentation required to read the SPICE Kernels and use the data contained therein to help plan observations or interpret space science data. This software and associated documentation are collectively called the NAIF Toolkit.

Kernel files are the major components of the SPICE system. The EPHEMERIS KERNEL_TYPE (SPK) contains spacecraft and planet, satellite or other target body ephemeris data that provide position and velocity of a spacecraft as a function of time. The TARGET_CONSTANTS KERNEL_TYPE (PCK) contains planet, satellite, comet, or asteroid cartographic constants for that object. The INSTRUMENT KERNEL_TYPE (IK) contains a collection of science instrument information, including specification of the mounting alignment, internal timing, and other information needed to interpret measurements made with the instrument. The POINTING KERNEL_TYPE (CK) contains pointing data (e.g., the inertially referenced attitude for a spacecraft structure upon which instruments are mounted, given as a function of time). The EVENTS KERNEL_TYPE (EK) contains event information (e.g, spacecraft and instrument commands, ground data system event logs, and experimenter's notebook comments). The LEAPSECONDS KERNEL_TYPE (LSK) contains an account of the leapseconds needed to correlate civil time (UTC or GMT) with ephemeris time (TDB). This is the measure of time used in the SP kernel files. The Spacecraft Clock coefficients kernel (SCLK) contains the data needed to correlate a spacecraft clock with ephemeris time.

Data products referencing a particular SPICE kernel would do so through the SOURCE_PRODUCT_ID keyword in their label with the value corresponding to that of the PRODUCT_ID within the SPICE_KERNEL label. The PRODUCT_ID keyword is unique to a data product.

Required Keywords

1. DESCRIPTION
2. INTERCHANGE_FORMAT
3. KERNEL_TYPE

Optional Keywords

None

Required Objects

None

Optional Objects

None

Example

NOTE: The following example of a SPICE CK (Pointing) Kernel attached label may have been modified to reflect current PDS standards and is not intended to contain actual PDS ingested values. You will notice that some label information is actually inside the Kernel file which allows NAIF tools to extract information to produce the PDS label.

```
CCSD...
PDS_VERSION_ID                          = PDS3
RECORD_TYPE                             = STREAM
MISSION_NAME                            = MARS_OBSERVER
SPACECRAFT_NAME                         = MARS_OBSERVER
DATA_SET_ID                             = “MO-M-SPICE-6-CK-V1.0”
FILE_NAME                               = "NAF0000D.TC"
PRODUCT_ID                              = "NAF0000D-CK"
PRODUCT_CREATION_TIME                   = 1992-04-14T12:00:00
PRODUCER_ID                             = "NAIF"
MISSION_PHASE_TYPE                      = "ORBIT"
PRODUCT_VERSION_TYPE                    = "TEST"
START_TIME                              = 1994-01-06T00:00:00
STOP_TIME                               = 1994-02-04T23:55:00
SPACECRAFT_CLOCK_START_COUNT             = "3/76681108.213"
SPACECRAFT_CLOCK_STOP_COUNT              = "4/79373491.118"
TARGET_NAME                             = MARS
INSTRUMENT_NAME                         = "MARS OBSERVER SPACECRAFT"
INSTRUMENT_ID                           = MO
SOURCE_PRODUCT_ID                       =
{"NAF0000C.BSP","NAF0000C.TLS","NAF0000C.TSC"}
NOTE                                    = "BASED ON EPHEMERIS IN NAF0000C.BSP. FOR SOFTWARE
TESTING ONLY."
OBJECT                                  = SPICE_KERNEL
INTERCHANGE_FORMAT                      = ASCII
KERNEL_TYPE                             = POINTING
DESCRIPTION                             = "This is a SPICE kernel file, designed to be accessed using NAIF Toolkit
software. Contact your flight project representative or the NAIF node of the Planetary Data System if you wish to obtain a copy of
the NAIF Toolkit.  The Toolkit consists of portable FORTRAN 77 code and extensive user documentation."
END_OBJECT                              = SPICE_KERNEL
END
CCSD...
_____________________
INTERNAL SPICE LABEL
SPICE DATA
```

A.27 TABLE

TABLEs are the natural storage format for collections of data from many instruments. They are also the most effective way of storing much of the meta-data which are used to identify and describe instrument observations.

The TABLE object is a uniform collection of rows containing ASCII or binary values stored in columns. The ROWS and COLUMNS of the TABLE object provide a natural correspondence to the records and fields often defined in interface specifications for existing data products. The value to use for the COLUMNS keyword in a TABLE object should be the actual number of COLUMN objects defined in the label. The INTERCHANGE_FORMAT keyword is used to distinguish between ASCII and binary table values.

ASCII vs. BINARY formats

ASCII tables provide the most portable format for access across a wide variety of computer platforms. They are also easily imported into a number of database management systems and spreadsheet applications. For these reasons, the PDS recommends the use of ASCII table formats whenever possible for archive products.

ASCII formats are generally less efficient for storing large quantities of data. In addition, raw or minimally processed data products and many pre-existing data products undergoing restoration are only available in binary formats.Where conversion to an ASCII format is neither cost effective nor desirable, BINARY table formats can be used.

Required Keywords

1. INTERCHANGE_FORMAT
2. ROWS
3. COLUMNS
4. ROW_BYTES

Optional Keywords

1. NAME
2. DESCRIPTION
3. ROW_PREFIX_BYTES
4. ROW_SUFFIX_BYTES
5. TABLE_STORAGE_TYPE

Required Objects

1. COLUMN

Optional Objects

1. CONTAINER

Many variations of the TABLE object are possible with the addition of the “optional” keywords and/or objects to the basic TABLE definition. While PDS supports these options, they are often not the best choices for archival data products. Recommended ASCII and binary table formats are provided in the following sections (A.27.1, A.27.2) with examples. Section A.27.3 provides examples of several TABLE variations and their possible application. Section A.27.4 provides specific guidelines for SPARE columns or unused fields within a TABLE.

A.27.1 Recommended ASCII TABLE Format

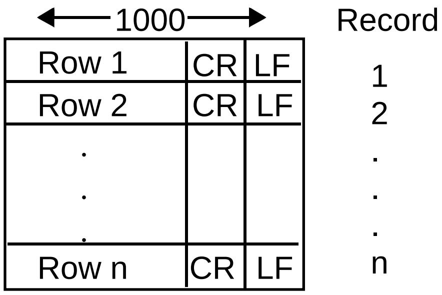
The recommended PDS table format uses ASCII COLUMN values, with a fixed size for each COLUMN. Each RECORD within the table is the same length and is terminated with a carriage-return/line-feed <CR><LF> pair. COLUMNs are separated by commas and character fields are enclosed in QUOTATION MARKS ("). The QUOTATION MARKs should surround the maximum COLUMN width. For example, a twelve character COLUMN called SPACECRAFT_NAME would be represented in the table as:

"VOYAGER 1   ", instead of "VOYAGER 1"

Numeric fields are right-justified in the allotted space and character fields are left-justified and blank padded on the right. This table format can be imported into many data management systems such as DBASE, FoxBase, Paradox, and Britton-Lee and into EXCEL spreadsheets.

The following label subset and illustration provide the general characteristics of a PDS recommended ASCII table with 1000 byte records:

```
RECORD_TYPE  = FIXED_LENGTH
RECORD_BYTES = 1000
...
OBJECT = TABLE
  INTERCHANGE_FORMAT = ASCII
  ROW_BYTES = 1000
  ...
END_OBJECT = TABLE
```



Example - Recommended ASCII TABLE

The following example is an ASCII index table with fixed length 71 byte records. Note that for ASCII tables, the delimiters (i.e., double quotes, commas, and line terminators <CR><LF>) are included in the byte count for each record (RECORD_BYTES). In this example, the delimiters are also included in the byte count for each row (ROW_BYTES). The <CR><LF> characters have been placed in columns 70 and 71.

Contents of file "INDEX.TAB"

```
------------------------------------------------------------------------
"F-MIDR ","F-MIDR.40N286;1 ","C", 42, 37,289,282,"F40N286/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.20N280;1 ","C", 22, 17,283,277,"F20N280/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.20N286;1 ","C", 22, 17,289,283,"F20N286/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.00N279;1 ","R",  2, -2,281,275,"F00N279/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.05N290;1 ","C",  7,  2,292,286,"F05N290/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.05S279;1 ","R", -2, -7,281,275,"F05S279/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.10S284;1 ","C", -7,-12,287,281,"F10S284/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.10S290;1 ","R", -7,-12,292,286,"F10S290/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.15S283;1 ","R",-12,-17,286,279,"F15S283/FRAME.LBL "<CR><LF>
"F-MIDR ","F-MIDR.15S289;1 ","R",-12,-17,291,285,"F15S289/FRAME.LBL "<CR><LF>
```

Contents of file "INDEX.LBL"

```
------------------------------------------------------------------------
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                = PDS3
RECORD_TYPE                   = FIXED_LENGTH
RECORD_BYTES                  = 71
FILE_RECORDS                  = 10
^INDEX_TABLE                  = "INDEX.TAB"

DATA_SET_ID                   = "MGN-V-RDRS-5-MIDR-FULL-RES-V1.0"
VOLUME_ID                     = MG_7777
PRODUCT_ID                    = "FMIDR.XYZ"
SPACECRAFT_NAME               = MAGELLAN
INSTRUMENT_NAME               = "RADAR SYSTEM"
TARGET_NAME                   = VENUS
PRODUCT_CREATION_TIME         = "N/A"
MISSION_PHASE_NAME            = PRIMARY_MISSION
NOTE                          = "This table lists all MIDRs on this volume.  It also includes the latitude and
longitude range for each MIDR and the directory in which it is found."

OBJECT                        = INDEX_TABLE
INTERCHANGE_FORMAT            = ASCII
ROWS                          = 10
COLUMNS                       = 8
ROW_BYTES                     = 71
INDEX_TYPE                    = SINGLE

OBJECT                        = COLUMN
NAME                          = PRODUCT_TYPE
DESCRIPTION                   = "Magellan DMAT type code.  Possible values are F-MIDR, C1-MIDR, C2-
                                MIDR, C3-MIDR, and P-MIDR."
DATA_TYPE                     = CHARACTER
```

```
START_BYTE                       = 2
BYTES                            = 7
END_OBJECT                       = COLUMN

OBJECT                           = COLUMN
NAME                             = PRODUCT_ID
DESCRIPTION                      = "Magellan DMAT name of product.
                                 Example:  F-MIDR.20N334;1"
DATA_TYPE                        = CHARACTER
START_BYTE                       = 12
BYTES                            = 16
END_OBJECT                       = COLUMN

OBJECT                           = COLUMN
NAME                             = SEAM_CORRECTION_TYPE
DESCRIPTION                      = "A value of C indicates that cross- track seam correction has been applied.  A
value of R indicates that the correction has not been applied."
DATA_TYPE                        = CHARACTER
START_BYTE                       = 31
BYTES                            = 1
END_OBJECT                       = COLUMN

OBJECT                           = COLUMN
NAME                             = MAXIMUM_LATITUDE
DESCRIPTION                      = "Northernmost frame latitude rounded to the nearest degree."
DATA_TYPE                        = INTEGER
UNIT                             = DEGREE
START_BYTE                       = 34
BYTES                            = 3
END_OBJECT                       = COLUMN

OBJECT                           = COLUMN
NAME                             = MINIMUM_LATITUDE
DESCRIPTION                      = "Southernmost frame latitude rounded to the nearest degree."
DATA_TYPE                        = INTEGER
UNIT                             = DEGREE
START_BYTE                       = 38
BYTES                            = 3
END_OBJECT                       = COLUMN

OBJECT                           = COLUMN
NAME                             = EASTERNMOST_LONGITUDE
DESCRIPTION                      = "Easternmost frame longitude rounded to the nearest degree."
DATA_TYPE                        = INTEGER
UNIT                             = DEGREE
START_BYTE                       = 42
BYTES                            = 3
END_OBJECT                       = COLUMN

OBJECT                           = COLUMN
NAME                             = WESTERNMOST_LONGITUDE
DESCRIPTION                      = "Westernmost frame longitude rounded to the nearest degree."
DATA_TYPE                        = INTEGER
UNIT                             = DEGREE
START_BYTE                       = 46
BYTES                            = 3
END_OBJECT                       = COLUMN

OBJECT                           = COLUMN
```

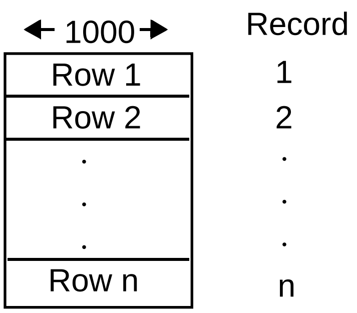
```
NAME                                = FILE_SPECIFICATION_NAME
DESCRIPTION                         = "Path and file name of frame table relative to CD-ROM root directory."
DATA_TYPE                           = CHARACTER
START_BYTE                          = 51
BYTES                               = 18
END_OBJECT                          = COLUMN

END_OBJECT                          = INDEX_TABLE
END
```

A.27.2 Recommended BINARY TABLE Format

The recommended PDS binary table format uses FIXED_LENGTH records, with each row of the table occupying a complete physical record (i.e. RECORD_BYTES = ROW_BYTES). This recommended format also discourages the use of BIT_COLUMN objects within COLUMNS in binary tables, primarily for portability reasons. Whenever possible, bit fields should be unpacked into more portable byte oriented COLUMNS. Unused bytes embedded within the binary table should be explicitly identified with COLUMNs named “SPARE” for completeness and to facilitate automated validation of these table structures.

The following label subset and illustration provide the general characteristics of a PDS recommended binary table with 1000 byte records:

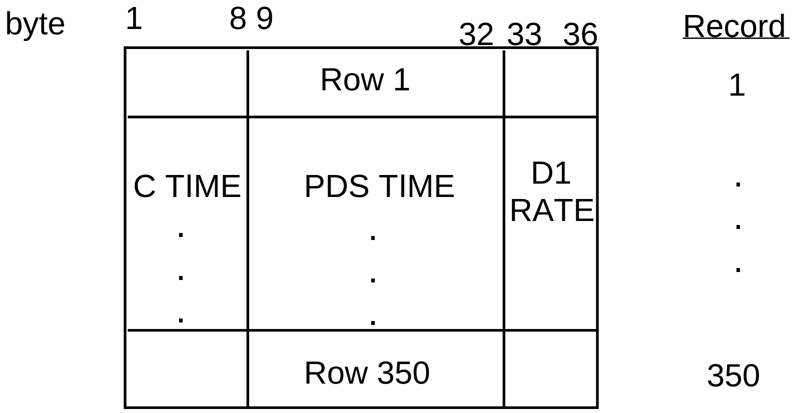
```
RECORD_TYPE  = FIXED_LENGTH
  RECORD_BYTES = 1000
  ...
  OBJECT = TABLE
    INTERCHANGE_FORMAT = BINARY
    ROW_BYTES = 1000
    ...
  END_OBJECT = TABLE
```



Example - Recommended Binary TABLE

The following is an example of a binary table consisting of 3 columns of data. The first two columns provide TIME information in both the PDS standard UTC format and an alternate format. The third column provides uncalibrated instrument measurements for the given times. This table could also be represented as a TIME_SERIES by the addition of sampling parameter keywords to describe the variation between each row of the table. The following illustration shows the layout and contents of the binary table in file "T890825.DAT".   The detached label file, "T890825.LBL" provides the complete description.

Contents of file "T890825.DAT":



Contents of file "T890825.LBL":

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                       = PDS3

/* File Characteristic Keywords */
RECORD_TYPE                          = FIXED_LENGTH
RECORD_BYTES                         = 36
FILE_RECORDS                         = 350
HARDWARE_MODEL_ID                    = "SUN SPARC STATION"
OPERATING_SYSTEM_ID                  = "SUN OS 4.1.1"

/* Data Object Pointers */
^TABLE                               = "T890825.DAT"

/* Identification Keywords */
DATA_SET_ID                          = "VG2-N-CRS-4-SUMM-D1-96SEC-V1.0"
SPACECRAFT_NAME                      = "VOYAGER 2"
INSTRUMENT_NAME                      = "COSMIC RAY SYSTEM"
TARGET_NAME                          = NEPTUNE
START_TIME                           = 1989-08-25T00:00:00.000Z
STOP_TIME                            = 1989-08-25T09:58:02.000Z
MISSION_PHASE_NAME                   = "NEPTUNE ENCOUNTER"
PRODUCT_ID                           = "T890825.DAT"
PRODUCT_CREATION_TIME                = "UNK"
SPACECRAFT_CLOCK_START_COUNT         = "UNK"
SPACECRAFT_CLOCK_STOP_COUNT          = "UNK"

/* Data Object Descriptions */
OBJECT                               = TABLE
INTERCHANGE_FORMAT                   = BINARY
ROWS                                 = 350
COLUMNS                              = 3
ROW_BYTES                            = 36
^STRUCTURE                           = "CRSDATA.FMT"
END_OBJECT                           = TABLE
END
```

Contents of file "CRSDATA.FMT":

```
------------------------------------------------------------------------
OBJECT                              = COLUMN
NAME                                = "C TIME"
UNIT                                = "SECONDS"
DATA_TYPE                           = REAL
START_BYTE                          = 1
BYTES                               = 8
MISSING                             = 1.0E+32
DESCRIPTION                         = "
Time column. This field contains time in seconds after Jan 01, 1966 but is displayed in the default time format selected by the user."
END_OBJECT                          = COLUMN

OBJECT                              = COLUMN
NAME                                = "PDS TIME"
UNIT                                = "TIME"
DATA_TYPE                           = CHARACTER
 START_BYTE                         = 9
BYTES                               = 24
DESCRIPTION                         = "
Date/Time string of the form yyyy-mm-ddThh:mm:ss.sss such that the representation of the date Jan 01, 2000  00:00:00.000 would
be 2000-01-01T00:00:00.000Z (Z indicates Universal Time)."
END_OBJECT                          = COLUMN

OBJECT                              = COLUMN
NAME                                = "D1 RATE"
UNIT                                = "COUNTS"
DATA_TYPE                           = "REAL"
START_BYTE                          = 33
BYTES                               = 4
MISSING                             = 1.0E+32
DESCRIPTION                         = "
The D1 rate is approximately porportional to the omnidirectional flux of electrons with kinetic energy > ~1MeV. To obtain greater
accuracy, the D1 calibration tables (see catalog) should be applied."
END_OBJECT                          = COLUMN
```

A.27.3 TABLE Variations

This section addresses a number of structural variations of “table based” data objects. As the structure of SERIES and SPECTRUM objects are similar and can be identical to the TABLE object, all three objects (TABLE, SERIES, and SPECTRUM) can be of the following structure types. The structural variations presented here are primarily due to the physical placement of the data (ROW_BYTES) in relation to the size of the data record (RECORD_BYTES), the type of the data (ASCII or BINARY), and the format of the data (FIXED_LENGTH or STREAM).

This section is not intended to be a complete reference for TABLE variations. Within the following examples, some illustrate a recommended data modelling approach, some illustrate alternate approaches, and other examples are included solely to document their existence.

Note: The examples in the following sections use OBJECT = TABLE, but OBJECT = SERIES or OBJECT = SPECTRUM could be substituted.

A.27.3.1 Record blocking in Fixed Length TABLES

The PDS recommended TABLE format requires the ROW_BYTES of the TABLE object to be equal to RECORD_BYTES of the file. This is not always the case, particularly when describing existing binary TABLE formats.

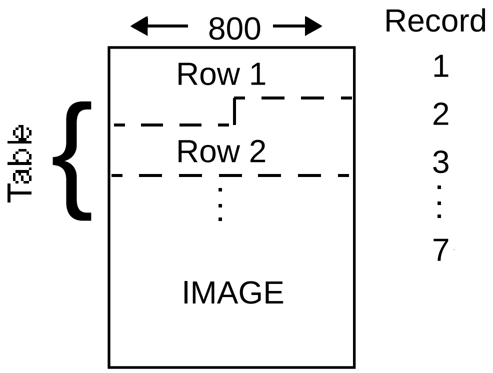
A common use of blocking occurs when two or more data objects are packaged into the same file, each requiring a different size record. In addition, rows in a TABLE are sometimes blocked into larger physical records to minimize input/output operations.

Rows in both ASCII or binary tables can be either larger or smaller than the physical record size specified by the RECORD_BYTES keyword.

Example - Binary Table with ROW_BYTES > RECORD_BYTES

The following label subset and illustration provide the general characteristics of a product containing an 800 byte IMAGE object together with a TABLE with 1200 byte rows:

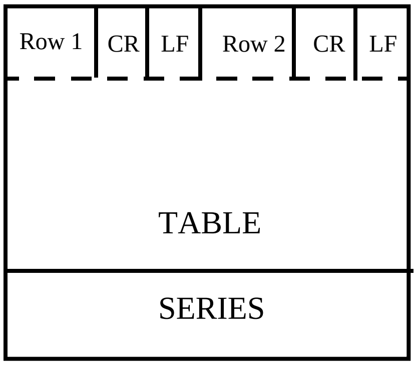
```
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 800
^TABLE                         =("IMAGE.IMG",1)
^IMAGE                         =("IMAGE.IMG",7)
  ...
OBJECT                         = TABLE
INTERCHANGE_FORMAT             = BINARY
ROW_BYTES                      = 1200
   ...
END_OBJECT                     = TABLE

OBJECT                         = IMAGE
SAMPLES                        = 800
SAMPLE_BITS                    = 8
   ...
END_OBJECT                     = IMAGE
```



Example - ASCII Table with ROW_BYTES < RECORD_BYTES

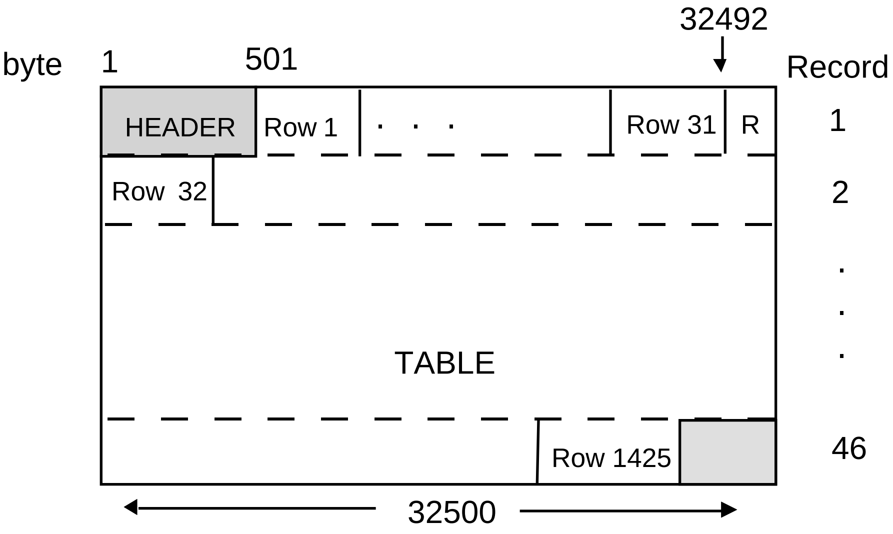
The following label subset and illustration provide the general characteristics of a product containing a SERIES object containing 800 byte rows together with a TABLE object with 400 byte rows:

```
RECORD_TYPE                = FIXED_LENGTH
RECORD_BYTES               = 800
...

OBJECT                     = TABLE
INTERCHANGE_FORMAT         = ASCII
ROW_BYTES                  =400
...
END_OBJECT                 = TABLE

OBJECT                     =SERIES
INTERCHANGE_FORMAT         =ASCII
ROW_BYTES                  = 800
...
END_OBJECT                 = SERIES
```



Example - Binary Table with ROW_BYTES < RECORD_BYTES

The following label subset and illustration provide the general characteristics of a product containing an HEADER object containing one 500 byte row together with a TABLE with 1032 byte rows. In this case, both the HEADER and TABLE rows are blocked into 32500 byte records. Note that the rows cross record boundaries.

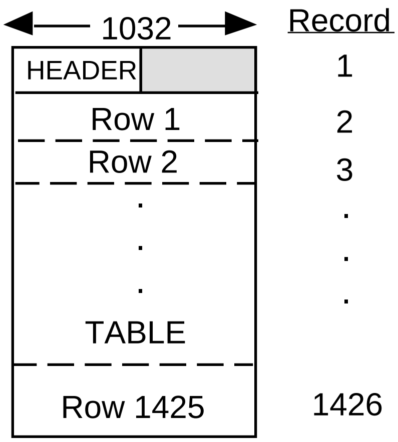


```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                    = PDS3

/* FILE CHARACTERISTICS */
RECORD_TYPE                       = FIXED_LENGTH
RECORD_BYTES                      = 32500
FILE_RECORDS                      = 46
^HEADER                           = ("ADF01141.3",1)
^TABLE                            = ("ADF01141.3",501<BYTES>)

/* IDENTIFICATION KEYWORDS */
DATA_SET_ID                       = "MGN-V-RDRS-5-CDR-ALT/RAD-V1.0"
PRODUCT_ID                        = "ADF01141.3"
TARGET_NAME                       = VENUS
SPACECRAFT_NAME                   = MAGELLAN
INSTRUMENT_NAME                   = "RADAR SYSTEM"
MISSION_PHASE_NAME                = PRIMARY_MISSION
PRODUCT_CREATION_TIME             = 1991-07-23T06:16:02.000Z
ORBIT_NUMBER                      = 1141
START_TIME                        = UNK
STOP_TIME                         = UNK
SPACECRAFT_CLOCK_START_COUNT      = UNK
SPACECRAFT_CLOCK_STOP_COUNT       = UNK
HARDWARE_VERSION_ID               = 01
SOFTWARE_VERSION_ID               = 02
UPLOAD_ID                         = M0356N
NAVIGATION_SOLUTION_ID            = "ID = M0361-12 "
DESCRIPTION                       = " This file contains binary records describing, in time order, each altimeter
footprint measured during an orbit of the Magellan radar mapper."

/* DATA OBJECT DEFINITION  DESCRIPTIONS */
OBJECT                            = HEADER
HEADER_TYPE                       = SFDU
BYTES                             = 500
END_OBJECT                            = HEADER
OBJECT                            = TABLE
INTERCHANGE_FORMAT                = BINARY
ROWS                              = 1425
COLUMNS                           = 40
ROW_BYTES                         = 1032
^STRUCTURE                        = "ADFTBL.FMT"
END_OBJECT                              = TABLE
END

Contents of format file "ADFTBL.FMT"
------------------------------------
OBJECT                            = COLUMN
NAME                              = SFDU_LABEL_AND_LENGTH
START_BYTE                        = 1
DATA_TYPE                         = CHARACTER
BYTES                             = 20
UNIT                              = "N/A"
DESCRIPTION                       = "
The SFDU_label_and_length element identifies the label and length of the Standard Format Data Unit (SFDU)."
END_OBJECT                        = COLUMN

OBJECT                            = COLUMN
NAME                              = FOOTPRINT_NUMBER
START_BYTE                        = 21
```

```
DATA_TYPE                    = LSB_INTEGER
BYTES                        = 4
UNIT                         = "N/A"
DESCRIPTION                  = "The footprint_number element provides a signed integer value. The altimetry
and radiometry processing program assigns footprint 0 to that observed at nadir at periapsis. The remaining footprints are located
along the spacecraft nadir track, with a separation that depends on the Doppler resolution of the altimeter at the epoch at which that
footprint is observed. Pre-periapsis footprints will be assigned negative numbers, post-periapsis footprints will be assigned positive
ones. A loss of several consecutive burst records from the ALT-EDR will result in missing footprint numbers."

END_OBJECT                   = COLUMN

...

OBJECT                       = COLUMN
 NAME                        = DERIVED_THRESH_DETECTOR_INDEX
 START_BYTE                  = 1001
 DATA_TYPE                   = LSB_UNSIGNED_INTEGER
 BYTES                       = 4
 UNIT                        = "N/A"
 DESCRIPTION                 = "The derived_thresh_detector_index element provides the value of the element
in range_sharp_echo_profile that satisfies the altimeter threshold detection algorithm, representing the distance to the nearest object
in this radar footprint in units of 33.2 meters, modulus a 10.02 kilometer altimeter range ambiguity."
END_OBJECT                   = COLUMN
```

Example - Alternate format; PDS Recommended

The following label subset and illustration provide an alternate data organization for the preceding example. In this example, a record size of 1032 is used to match the row size of the TABLE, and the 500 byte HEADER uses only a portion of the first record. This organization would conform to the PDS recommended TABLE structure.

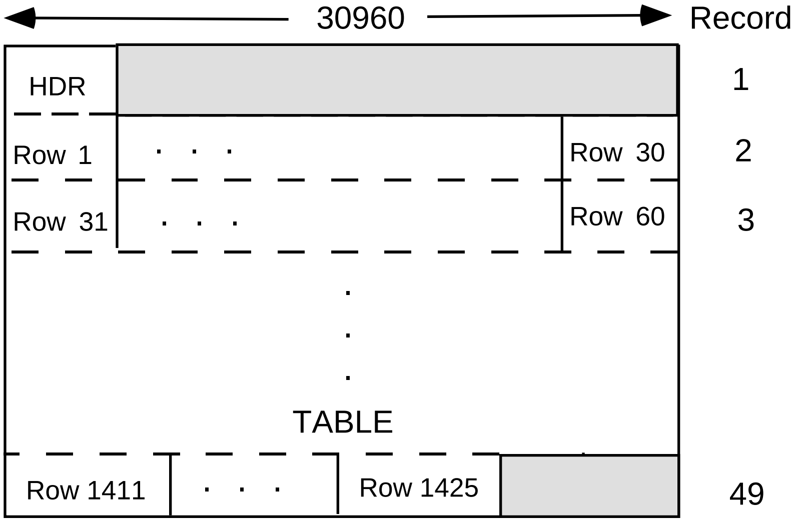


```
...
RECORD_TYPE                  = FIXED_LENGTH
RECORD_BYTES                 = 1032
FILE_RECORDS                 = 1426
^HEADER                      = ("ADF01141.3",1)
^TABLE                       = ("ADF01141.3",2)
```

```
 ...

/* DATA OBJECT DEFINITIONS */
OBJECT                                  = HEADER
 HEADER_TYPE                            = SFDU
 BYTES                                  = 500
END_OBJECT
OBJECT                                  = TABLE
 INTERCHANGE_FORMAT                     = BINARY
 ROWS                                   = 1425
 COLUMNS                                = 40
 ROW_BYTES                              = 1032
 ^STRUCTURE                             = "ADFTBL.FMT"
END_OBJECT
END
```

Example - Alternate format; Rows on Record Boundaries

The following label subset and illustration provide a second alternate data organization for the preceding example. In this example, a record size of 66048 is used to hold 30 rows of the TABLE. Again the 500 byte HEADER uses only a portion of the first record.



```
...
RECORD_TYPE                             = FIXED_LENGTH
RECORD_BYTES                            =30960
FILE_RECORDS                            =49
^HEADER                                 = ("ADF01141.3",1)
^TABLE                                  = ("ADF01141.3",2)
 ...

/* DATA OBJECT DEFINITIONS */
OBJECT                                  = HEADER
 HEADER_TYPE                            = SFDU
 BYTES                                  = 500
END_OBJECT
```

```
OBJECT                         = TABLE
 INTERCHANGE_FORMAT            = BINARY
 ROWS                          = 1425
 COLUMNS                       = 40
 ROW_BYTES                     = 1032
 ^STRUCTURE                    = "ADFTBL.FMT"
END_OBJECT
END
```

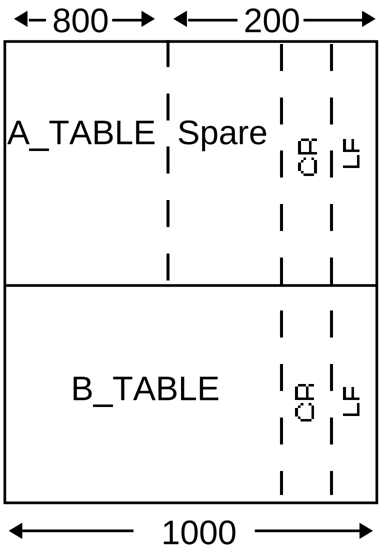
A.27.3.2 Multiple TABLEs with varying ROW_BYTES

A data product may contain several ASCII or binary tables, each with a different row size.

Example - Fixed Length Records - Multiple ASCII tables

The following label subset and illustration utilizes fixed length records of the maximum row size. The smaller table is padded with spares preceding the <CR><LF>. Note that the ROW_BYTES keyword in A_TABLE could be replaced by ROW_BYTES = 800 and ROW_SUFFIX_BYTES = 200. See section A.27.4 for further information on handling spares.

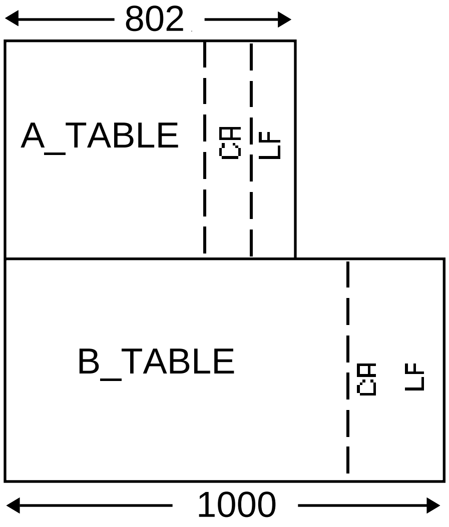
```
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 1000
    ...
OBJECT                         = A_TABLE
INTERCHANGE_FORMAT             = ASCII
ROW_BYTES                      = 1000
      ...
END_OBJECT                     = A_TABLE

OBJECT                         = B_TABLE
INTERCHANGE_FORMAT             = ASCII
ROW_BYTES                      = 1000
      ...
END_OBJECT                     = B_TABLE
```



Example - Stream Records - Multiple ASCII tables

The following label subset and illustration utilizes stream records for the same data as the previous example, placing the <CR><LF> pair at the end of the data in each table. There is no need to pad out the smaller table using the STREAM format, and the RECORD_BYTES keyword is not applicable.

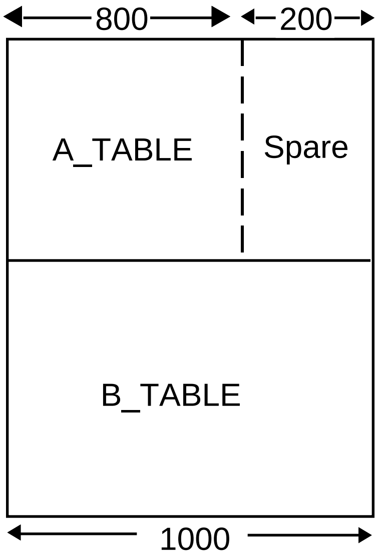
```
RECORD_TYPE                       = STREAM
  ...
OBJECT                            = A_TABLE
INTERCHANGE_FORMAT                = ASCII
ROW_BYTES                         = 802
  ...
END_OBJECT                        = A_TABLE

OBJECT                            = B_TABLE
INTERCHANGE_FORMAT                = ASCII
ROW_BYTES                         = 1000
  ...
  END_OBJECT = B_TABLE
```



Example - Fixed Length Records - Multiple Binary tables

The following label subset and illustration utilizes fixed length records of the maximum row size. The smaller table has a spare set of bytes in each record, explicitly defined in a “spare” COLUMN object. Note that the ROW_BYTES keyword in A_TABLE could be replaced by ROW_BYTES = 800 and ROW_SUFFIX_BYTES = 200, instead of explicitly defining the SPARE column. See section A.27.4 for further information on handling spares.

```
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 1000
   ...
OBJECT                         = A_TABLE
INTERCHANGE_FORMAT             = BINARY
ROW_BYTES                      = 1000
    ...
OBJECT                         = COLUMN
NAME                           = "SPARE"
DATA_TYPE                      = "N/A"
START_BYTE                     = 801
BYTES                          = 200
END_OBJECT                     = COLUMN
END_OBJECT                     = A_TABLE

OBJECT                         = B_TABLE
INTERCHANGE_FORMAT             = BINARY
ROW_BYTES                      = 1000
   ...
END_OBJECT                     = B_TABLE
```



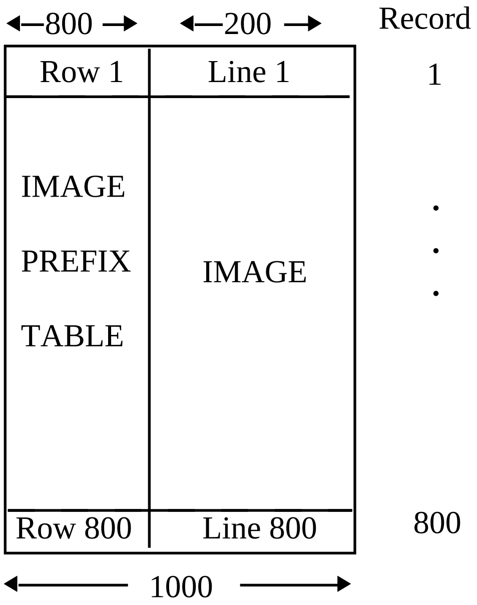
A.27.3.3 ROW_PREFIX or ROW_SUFFIX use

There are currently two methods to utilize ROW_PREFIX_BYTES and ROW_SUFFIX_BYTES in TABLE objects. The first application is limited to Binary TABLE objects that are adjacent to another object, such that each object shares the same record in a file. The second application is for identifying spare bytes at the beginning or end of a record that are not considered part of the TABLE data.

Example - Row Suffix use for compound TABLE and IMAGE

The following label subset and illustration utilizes fixed length records each containing a row of a TABLE data object, and a line of an IMAGE object. This is a common format for providing ancillary information applicable to each IMAGE line.

```
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 1000
  ...
OBJECT                         = TABLE
INTERCHANGE_FORMAT             = BINARY
ROW_BYTES                      = 200
ROW_SUFFIX_BYTES               = 800
  ...
END_OBJECT                     = TABLE

OBJECT                         = IMAGE
LINE_SAMPLES                   = 800
SAMPLE_BITS                    = 8
LINE_PREFIX_BYTES              = 200
  ...
END_OBJECT                     = IMAGE
```



The following RULES apply to the use of ROW_PREFIX_BYTES and ROW_SUFFIX_BYTES:

1. For compound “table based” objects (TABLE, SPECTRUM, SERIES) in a data product, or for identifying Spare parts of a record:

RECORD_BYTES = ROW_BYTES + ROW_PREFIX_BYTES + ROW_SUFFIX_BYTES

2. For compound “table based” and IMAGE objects in a data product:

RECORD_BYTES = (LINE_SAMPLES * SAMPLE_BITS / 8) + ROW_PREFIX_BYTES + ROW_SUFFIX_BYTES

A.27.3.4 CONTAINER Object use

Complicated or lengthy tables that have a set of COLUMNS that repeat are often easier to describe with an illustration and the use of the CONTAINER sub-object in a TABLE description. The use of the container sub-object eliminates the need for repeating a group of COLUMN objects and adjusting the START_BYTE locations and descriptions for each repetition. Section A.8 provides an example of a TABLE utilizing the CONTAINER sub-object.

A.27.4 Guidelines for SPARE fields

There is often a need to reserve SPARE (or pad, filler, etc.). bytes in TABLE, SPECTRUM, and SERIES objects. While this is not required, it facilitates validation and ensures that the data producer did not inadvertently forget to account for some fields in the data. These guidelines differ slightly for BINARY and ASCII tables and FIXED_LENGTH or STREAM record files.

In all of the following guidelines, “embedded spares” refer to empty or spare bytes that are currently unused and are not defined as part of a data COLUMN.

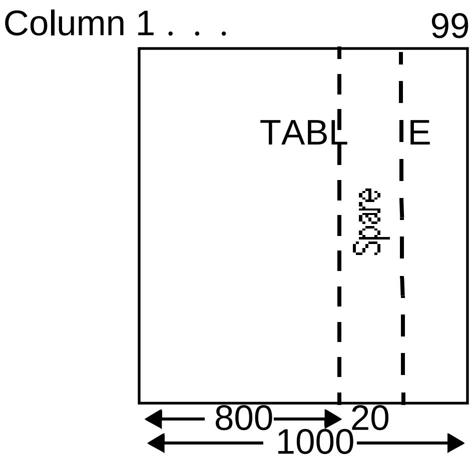
A.27.4.1 BINARY Tables - Fixed Length Records

The guidelines for handling SPARE fields in Fixed Length Binary Tables are:

- Embedded spares are allowed.
- Embedded spares are explicitly defined (with COLUMN Objects).
- Multiple Spare columns may all have NAME = SPARE
- Spares are allowed at the beginning or end of each row of data.
- Spares at the beginning or end of the data can be identified with
 1) an explicit COLUMN object or
or
 2) use of ROW_PREFIX_BYTES or ROW_SUFFIX_BYTES (note that these bytes should not be included in the value of ROW_BYTES)
- DATA_TYPE for Spare COLUMNS in binary table is 'N/A'

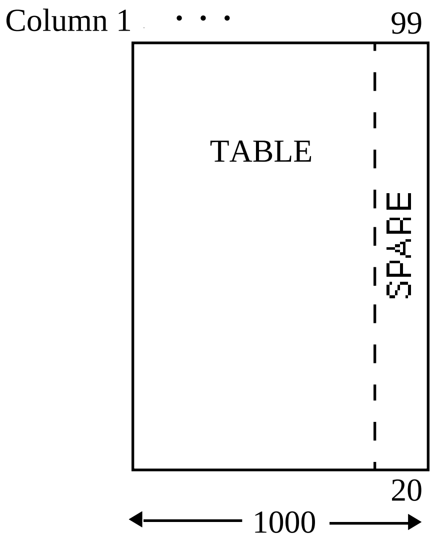
Example - SPARE field embedded in a Binary TABLE

```
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 1000
  ...
OBJECT                         = TABLE
INTERCHANGE_FORMAT             = BINARY
ROW_BYTES                      = 1000
COLUMNS                        = 99
    ...
OBJECT                         = COLUMN
NAME                           = SPARE
START_BYTE                     = 801
BYTES                          = 20
DATA_TYPE                      = "N/A"
     ...
END_OBJECT                     = COLUMN
END_OBJECT                     = TABLE
```



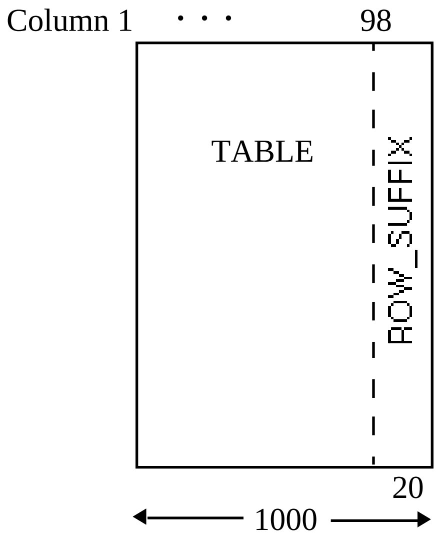
Example - Spares at end of a Binary TABLE - Explicit 'SPARE' Column

```
RECORD_TYPE                          = FIXED_LENGTH
RECORD_BYTES                         = 1000
  ...
OBJECT                               = TABLE
INTERCHANGE_FORMAT                   = BINARY
ROW_BYTES                            = 1000
COLUMNS                              = 99
   ...
OBJECT                               = COLUMN
NAME                                 = SPARE
BYTES                                = 20
DATA_TYPE                            = "N/A"
START_BYTE                           = 981
    ...
END_OBJECT                           = COLUMN
END_OBJECT                           = TABLE
```



Example - Spares at end of a Binary TABLE - ROW_SUFFIX use

```
RECORD_TYPE                          = FIXED_LENGTH
RECORD_BYTES                         = 1000
  ...
OBJECT                               = TABLE
INTERCHANGE_FORMAT                   = BINARY
ROW_BYTES                            = 980
ROW_SUFFIX_BYTES                     = 20
COLUMNS                              = 98
  ...
END_OBJECT                           = TABLE
```



A.27.4.2 ASCII Tables - Fixed Length Records

In ASCII tables, field delimiters (") and (,) and the <CR><LF> pair are considered part of the data, even though the COLUMN objects attributes do not include them. Spares in ASCII tables are limited to the “space” character (ASCII 20). The guidelines for handling SPARE fields in Fixed Length ASCII Tables are:

- Embedded spares are not allowed.
- Spares are allowed at the end of each row of data.
- The <CR><LF> follows the spare data.
- There are no delimiters (commas or quotes) surrounding the spares.
- Spares at the end of the data can be ignored (like field delimiters and CR LF) or they can be identified

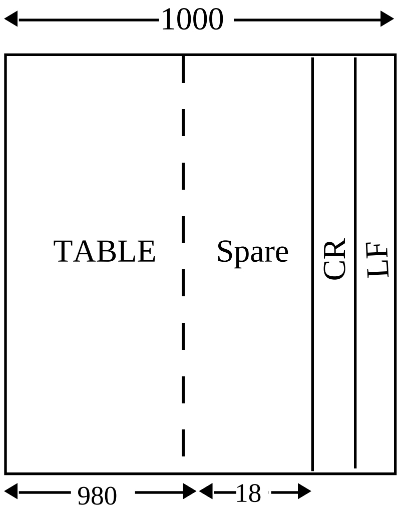
1) in the Table Description

or

2) by using ROW_SUFFIX_BYTES (note that these bytes should not be included in the value of ROW_BYTES)

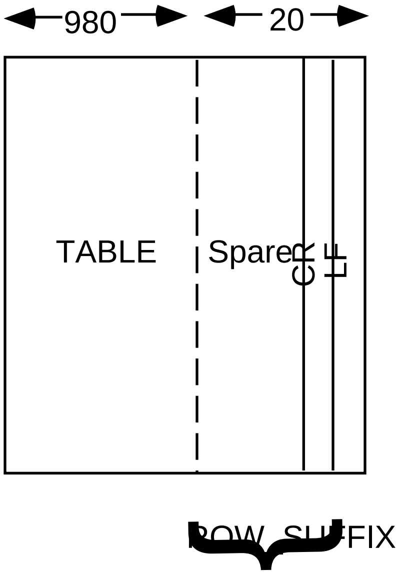
Example - SPARE field at end of ASCII TABLE - Table description note

```
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 1000
...
OBJECT                         = TABLE
INTERCHANGE_FORMAT             = ASCII
ROW_BYTES                      = 1000
...


DECRIPTION                     ="This table contains
980 bytes of table data followed by 18 bytes of blank spares.
Byte 999 and 1000 contain the <CR> <LF>
pair."
```



Example - Spares at end of a ASCII TABLE - ROW_SUFFIX use.

```
RECORD_TYPE                          = FIXED_LENGTH
RECORD_BYTES                         = 1000
...
OBJECT                               = TABLE
INTERCHANGE_FORMAT                   = ASCII
ROW_BYTES                            = 980
ROW_SUFFIX_BYTES                     = 20
...
END_OBJECT                           = TABLE
```



A.27.5 ASCII Tables - STREAM Records

Spares are not used with ASCII Tables in STREAM record formats. In STREAM files, the last data field explicitly defined with a COLUMN object is followed immediately by the <CR><LF> pair. Since there is no use for spares at the end of the data, and embedded spares are not allowed in ASCII tables, spares are not applicable here.

A.28 TEXT

The TEXT object contains plain text which begins immediately after the END statement. It is recommended that TEXT objects contain no special formatting characters, with the exception of the carriage return/line feed sequence and the page break. Tabs are discouraged, since they are interpreted differently by different programs. It is important to include BOTH the carriage return and line feed characters when preparing files for use on a variety of host systems.

Use of the Macintosh or Unix line terminators will cause text to be unreadable on other host computers. It is recommended that text lines be limited to 80 characters inclusive of the Carriage Return (Control M, HexOxOd) and Line Feed (Control J, HexOxOa) line delimiters.

NOTE: The text object is used in files describing the contents of an archive volume or the contents of a directory, such as AAREADME.TXT, DOCINFO.TXT, VOLINFO.TXT, SOFTINFO.TXT, etc. These files must be in plain unmarked ASCII text and always have the file name extension of .TXT. Documents placed on the volume in plain ASCII text, on the other hand, must be described using the DOCUMENT object. (See the definition of the DOCUMENT Object in Appendix A.)

The NOTE field provides a brief introduction to the TEXT.

Required Keywords

1. NOTE
2. PUBLICATION_DATE

Optional Keywords

1. INTERCHANGE_FORMAT

Required Objects

None

Optional Objects

None

Example

The example below is a portion of an AAREADME.TXT file.

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                          = PDS3
RECORD_TYPE                             = STREAM

OBJECT                                  = TEXT
PUBLICATION_DATE                        = 1991-05-28
NOTE                                    = "Introduction to this CD-ROM volume."
END_OBJECT                              = TEXT
END
        GEOLOGIC REMOTE SENSING FIELD EXPERIMENT

This set of compact read-only optical disks (CD-ROMs) contains a data collection acquired by ground-based and airborne
instruments during the Geologic Remote Sensing Field Experiment (GRSFE).  Extensive documentation is also included.  GRSFE
took place in July, September, and October, 1989, in the southern Mojave Desert, Death Valley, and the Lunar Crater Volcanic
Field, Nevada.  The purpose of these CD-ROMs is to make available in a compact form through the Planetary Data System (PDS)
a collection of relevant data to conduct analyses in preparation for the Earth Observing System (EOS), Mars Observer (MO), and
other missions.  The generation of this set of CD-ROMs was sponsored by the NASA Planetary Geology and Geophysics Program,
the Planetary Data System (PDS) and the Pilot Land Data System (PLDS).

This AAREADME.TXT file is one of the two nondirectory files located in the top level directory of each CD-ROM volume in this
collection. The other file, VOLDESC.CAT, contains an overview of the data sets on these CD-ROMs and is written in a format that
is designed for access by computers.  These two files appear on every volume in the collection.  All other files on the CD-ROMs
are located in directories below the top level directory ....
```

A.29 VOLUME

The VOLUME object describes a physical or logical unit used to store or distribute data products (e.g. a magnetic tape, CD-ROM disk, On-Line Magnetic disk or floppy disk) which contain directories and files. The directories and files may include documentation, software, calibration and geometry information as well as the actual science data.

Required Keywords

1. DATA_SET_ID
2. DESCRIPTION
3. MEDIUM_TYPE
4. PUBLICATION_DATE
5. VOLUME_FORMAT
6. VOLUME_ID
7. VOLUME_NAME
8. VOLUME_SERIES_NAME
9. VOLUME_SET_NAME
10.VOLUME_SET_ID
11.VOLUME_VERSION_ID
12.VOLUMES

Optional Keywords

1. BLOCK_BYTES
2. DATA_SET_COLLECTION_ID
3. FILES
4. HARDWARE_MODEL_ID
5. LOGICAL_VOLUMES
6. LOGICAL_VOLUME_PATH_NAME
7. MEDIUM_FORMAT
8. NOTE
9. OPERATING_SYSTEM_ID
10. PRODUCT_TYPE
11. TRANSFER_COMMAND_TEXT
12. VOLUME_INSERT_TEXT

Required Objects

1. CATALOG
2. DATA_PRODUCER

Optional Objects

1. DIRECTORY
2. FILE
3. DATA_SUPPLIER

Example 1 (Typical CD-ROM Volume)

Please see example in A.5 CATALOG.

Example 2 (Tape Volume)

The following VOLUME object example shows how directories and files are indicated when a volume is stored on ANSI tape for transfer. This form should be used when transferring volumes of data on media which do not support hierarchical directory structures (for example, submitting a volume of data for premastering). The VOLDESC.CAT file will contain the standard volume keywords, but the values of MEDIUM_TYPE, MEDIUM_FORMAT and VOLUME_FORMAT indicate that the volume is stored on tape.

In this example two files are defined in the root directory of the volume, VOLDESC.CAT and AAREADME.TXT. The first directory object defines the CATALOG directory which contains meta data in the High Level Catalog Templates. Here they all exist in one file, CATALOG.CAT. The second directory object defines an INDEX subdirectory, with three files embedded in it (INDXINFO.TXT, INDEX.LBL, INDEX.TAB). Following that directory, the first data directory is defined. Note that the sequence number field indicates the sequence of the file on the tape volume.

```
------------------------------------------------------------------------
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                    = PDS3
OBJECT                            = VOLUME
VOLUME_SERIES_NAME                = "MISSION TO MARS"
VOLUME_SET_NAME                   = "MARS DIGITAL IMAGE MOSAIC AND DIGITAL TERRAIN MODEL"
VOLUME_SET_ID                     = USA_NASA_PDS_VO_2001_TO_VO_2007
VOLUMES                           = 7
VOLUME_NAME                       = "MDIM/DTM VOLUME 7: GLOBAL COVERAGE"
VOLUME_ID                         = VO_2007
VOLUME_VERSION_ID                 = "VERSION 1"
PUBLICATION_DATE                  = 1992-04-01
DATA_SET_ID                       = "VO1/VO2-M-VIS-5-DTM-V1.0"
MEDIUM_TYPE                       = "8-MM HELICAL SCAN TAPE"
MEDIUM_FORMAT                     = "2 GB"
VOLUME_FORMAT                     = ANSI
HARDWARE_MODEL_ID                 = "VAX 11/750"
OPERATING_SYSTEM_ID               = "VMS 4.6"
DESCRIPTION                       = "This volume contains the Mars Digital Terrain Model and Mosaicked Digital
Image Model covering the entire planet at resolutions of 1/64 and 1/16 degree/pixel.  The volume also contains Polar Stereographic
projection files of the north and south pole areas from 80 to 90 degrees latitude; Mars Shaded Relief Airbrush Maps at 1/16 and 1/
4 degree/pixel; a gazetteer of Mars features; and a table of updated viewing geometry files of the Viking EDR images that comprise
the MDIM."
MISSION_NAME                      = VIKING
SPACECRAFT_NAME                   = {VIKING_ORBITER_1,VIKING_ORBITER_2}
SPACECRAFT_ID                     = {VO1,VO2}
```

```
OBJECT                              = DATA_PRODUCER
INSTITUTION_NAME                    = "U.S.G.S. FLAGSTAFF"
FACILITY_NAME                       = "BRANCH OF ASTROGEOLOGY"
FULL_NAME                           = "Eric M. Eliason"
DISCIPLINE_NAME                     = "IMAGE PROCESSING"
ADDRESS_TEXT                        = " Branch of Astrogeology \n
                                    United States Geological Survey\n
                                    2255 North Gemini Drive\n
                                     Flagstaff, Arizona. 86001 USA"
END_OBJECT                          = DATA_PRODUCER

OBJECT                              = CATALOG
^CATALOG                            = "CATALOG.CAT"
END_OBJECT                          = CATALOG

OBJECT                              = FILE
FILE_NAME                           = "VOLDESC.CAT"
RECORD_TYPE                         = STREAM
SEQUENCE_NUMBER                     = 1
END_OBJECT                          = FILE

OBJECT                              = FILE
FILE_NAME                           = "AAREADME.TXT"
RECORD_TYPE                         = STREAM
SEQUENCE_NUMBER                     = 2
END_OBJECT                          = FILE

OBJECT                              = DIRECTORY
NAME                                = CATALOG

OBJECT                              = FILE
FILE_NAME                           = "CATALOG.CAT"
RECORD_TYPE                         = STREAM
SEQUENCE_NUMBER                     = 3
END_OBJECT                          = FILE
END_OBJECT                          = DIRECTORY

OBJECT                              = DIRECTORY
NAME                                = DOCUMENT

OBJECT                              = FILE
FILE_NAME                           = "VOLINFO.TXT"
RECORD_TYPE                         = STREAM
SEQUENCE_NUMBER                     = 4
END_OBJECT                          = FILE

OBJECT                              = FILE
FILE_NAME                           = "DOCINFO.TXT"
RECORD_TYPE                         = STREAM
SEQUENCE_NUMBER                     = 5
END_OBJECT                          = FILE
END_OBJECT                          = DIRECTORY

OBJECT                              = DIRECTORY
NAME                                = INDEX

OBJECT                              = FILE
FILE_NAME                           = "INDXINFO.TXT"
```

```
RECORD_TYPE                    = STREAM
SEQUENCE_NUMBER                = 6
END_OBJECT                     = FILE

OBJECT                         = FILE
FILE_NAME                      = "INDEX.LBL"
RECORD_TYPE                    = STREAM
SEQUENCE_NUMBER                = 7
END_OBJECT                     = FILE

OBJECT                         = FILE
FILE_NAME                      = "INDEX.TAB"
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 512
FILE_RECORDS                   = 6822
SEQUENCE_NUMBER                = 8
END_OBJECT                     = FILE
END_OBJECT                     = DIRECTORY

OBJECT                         = DIRECTORY
NAME                           = MG00NXXX

OBJECT                         = FILE
FILE_NAME                      = "MG00N007.IMG"
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 964
FILE_RECORDS                   = 965
SEQUENCE_NUMBER                = 9
END_OBJECT                     = FILE

OBJECT                         = FILE
FILE_NAME                      = "MG00N012.IMG"
RECORD_TYPE                    = FIXED_LENGTH
RECORD_BYTES                   = 964
FILE_RECORDS                   = 965
SEQUENCE_NUMBER                = 10
END_OBJECT                     = FILE

END_OBJECT                     = DIRECTORY

END_OBJECT                     = VOLUME
END
```

Example 3a (CD-ROM Volume containing logical volumes)

Examples 3a and 3b illustrate the use of the VOLUME Object in the top level and at the logical volume level of an archive volume. Note that the VOLUME Object is required at both levels.

For examples 3a and 3b, the CD-ROM is structured as three separate logical volumes with root directories named PPS/, UVS/ and RSS/. An additional SOFTWARE directory is supplied at volume root for use with all logical volumes.

Example 3a illustrates the use of the VOLUME Object present at the top level of a CD-ROM containing logical volumes. Note usage of the keywords DATA_SET_ID, LOGICAL_VOLUMES, and LOGICAL_VOLUME_PATH_NAME.

```
PDS_VERSION_ID                          = PDS3
OBJECT = VOLUME
VOLUME_SERIES_NAME                      = "VOYAGERS TO THE OUTER PLANETS"
VOLUME_SET_NAME                         = "PLANETARY RING OCCULTATIONS FROM VOYAGER"
VOLUME_SET_ID                           = "USA_NASA_PDS_VG_3001"
VOLUMES                                 = 1
MEDIUM_TYPE                             = "CD-ROM"
VOLUME_FORMAT                           = "ISO-9660"
VOLUME_NAME                             = "VOYAGER PPS/UVS/RSS RING OCCULTATIONS"
VOLUME_ID                               = "VG_3001"
VOLUME_VERSION_ID                       = "VERSION 1"
PUBLICATION_DATE                        = 1994-03-01
DATA_SET_ID                             = {"VG2-SR/UR/NR-PPS-4-OCC-V1.0",
                                        "VG1/VG2-SR/UR/NR-UVS-4-OCC-V1.0","VG1/VG2-SR/UR/NR-RSS-4-
                                        OCC-V1.0"}
LOGICAL_VOLUMES                         = 3
LOGICAL_VOLUME_PATH_NAME                = {"PPS/", "UVS/", "RSS/"}
DESCRIPTION                             = "This volume contains the Voyager 1 and Voyager 2 PPS/UVS/RSS ring
occultation and ODR data sets. Included are data files at a variety of levels of processing, plus ancillary geometry, calibration and
trajectory files plus software and documentation.

This CD-ROM is structured as three separate logical volumes with root directories named PPS/, UVS/ and RSS/. An additional
SOFTWARE directory is supplied at volume root for use with all logical volumes."

OBJECT                                  = DATA_PRODUCER
INSTITUTION_NAME                        = "PDS RINGS NODE"
FACILITY_NAME                           = "NASA AMES RESEARCH CENTER"
FULL_NAME                               = "DR. MARK R. SHOWALTER"
DISCIPLINE_NAME                         = "RINGS"
ADDRESS_TEXT                            = "Mail Stop 245-3 \n
                                             NASA Ames Research Center \n
                                             Moffett Field, CA 94035-1000"
END_OBJECT                              = DATA_PRODUCER

OBJECT                                  = CATALOG
DATA_SET_ID                             = "VG2-SR/UR/NR-PPS-4-OCC-V1.0"
LOGICAL_VOLUME_PATH_NAME                = "PPS/"
^MISSION_CATALOG                        = "MISSION.CAT"
^INSTRUMENT_HOST_CATALOG                = "INSTHOST.CAT"
^INSTRUMENT_CATALOG                     = "INST.CAT"
^DATA_SET_COLLECTION_CATALOG            = "DSCOLL.CAT"
^DATA_SET_CATALOG                       = "DATASET.CAT"
^REFERENCE_CATALOG                       = "REF.CAT"
^PERSONNEL_CATALOG                      = "PERSON.CAT"
```

```
END_OBJECT                          = CATALOG

OBJECT                              = CATALOG
DATA_SET_ID                         = "VG1/VG2-SR/UR/NR-UVS-4-OCC-V1.0"
LOGICAL_VOLUME_PATH_NAME            = "UVS/"
^MISSION_CATALOG                    = "MISSION.CAT"
^INSTRUMENT_HOST_CATALOG            = "INSTHOST.CAT"
^INSTRUMENT_CATALOG                 = "INST.CAT"
^DATA_SET_COLLECTION_CATALOG        ="DSCOLL.CAT"
^DATA_SET_CATALOG                   = "DATASET.CAT"
^REFERENCE_CATALOG                  = "REF.CAT"
^PERSONNEL_CATALOG                  = "PERSON.CAT"
END_OBJECT                          = CATALOG

OBJECT                              = CATALOG
DATA_SET_ID                         = "VG1/VG2-SR/UR/NR-RSS-4-OCC-V1.0"
LOGICAL_VOLUME_PATH_NAME            = "RSS/"
^MISSION_CATALOG                    = "MISSION.CAT"
^INSTRUMENT_HOST_CATALOG            = "INSTHOST.CAT"
^INSTRUMENT_CATALOG                 = "INST.CAT"
^DATA_SET_COLLECTION_CATALOG        = "DSCOLL.CAT"
^DATA_SET_CATALOG                   = "DATASET.CAT"
^REFERENCE_CATALOG                  = "REF.CAT"
^PERSONNEL_CATALOG                  = "PERSON.CAT"
END_OBJECT                          = CATALOG

END_OBJECT                          = VOLUME
END
```

Example 3b (PPS/VOLDESC.CAT -- CD-ROM logical volume)

Example 3b illustrates the use of the Volume object which is required at the top level of a logical volume. Note the difference in values for the keywords DATA_SET_ID and LOGICAL_VOLUME_PATH_NAME from those used at the top level of the CD-ROM (example 3a). Also note that the keyword LOGICAL_VOLUMES does not appear here.

```
PDS_VERSION_ID                      = PDS3
OBJECT = VOLUME
VOLUME_SERIES_NAME                  = "VOYAGERS TO THE OUTER PLANETS"
VOLUME_SET_NAME                     = "PLANETARY RING OCCULTATIONS
                                    FROM VOYAGER"
VOLUME_SET_ID                       = "USA_NASA_PDS_VG_3001"
VOLUMES                             = 1
MEDIUM_TYPE                         = "CD-ROM"
VOLUME_FORMAT                       = "ISO-9660"
VOLUME_NAME                         = "VOYAGER PPS/UVS/RSS RING
                                    OCCULTATIONS"
VOLUME_ID                           = "VG_3001"
VOLUME_VERSION_ID                   = "VERSION 1"
PUBLICATION_DATE                    = 1994-03-01
DATA_SET_ID                         = "VG2-SR/UR/NR-PPS-4-OCC-V1.0"
LOGICAL_VOLUME_PATH_NAME            = "PPS/"
DESCRIPTION                         = "This logical volume contains the Voyager 2 PPS ring occultation data sets.
Included are data files at a variety of levels of processing, plus ancillary geometry, calibration  and trajectory files plus software and
documentation."

OBJECT                              = DATA_PRODUCER
```

```
INSTITUTION_NAME                    = "PDS RINGS NODE"
FACILITY_NAME                       = "NASA AMES RESEARCH CENTER"
FULL_NAME                           = "DR. MARK R. SHOWALTER"
DISCIPLINE_NAME                     = "RINGS"
ADDRESS_TEXT                        = "Mail Stop 245-3
NASA Ames Research Center
Moffett Field, CA 94035-1000"
END_OBJECT                          = DATA_PRODUCER

OBJECT                              = CATALOG
DATA_SET_ID                         = "VG2-SR/UR/NR-PPS-4-OCC-V1.0"
LOGICAL_VOLUME_PATH_NAME            = "PPS/"
^MISSION_CATALOG                    = "MISSION.CAT"
^INSTRUMENT_HOST_CATALOG            = "INSTHOST.CAT"
^INSTRUMENT_CATALOG                 = "INST.CAT"
^DATA_SET_COLLECTION_CATALOG        = "DSCOLL.CAT"
^DATA_SET_CATALOG                   = "DATASET.CAT"
^REFERENCE_CATALOG                  = "REF.CAT"
^PERSONNEL_CATALOG                  = "PERSON.CAT"
END_OBJECT                          = CATALOG

END_OBJECT                          = VOLUME
END
```

Appendix B

Complete PDS Catalog Object Template Set

This appendix provides a complete set of the PDS catalog objects in alphabetical order. Each section includes a description, a list of sub-objects, guidelines to follow in filling them out, and a specific example of the object.

The templates are used to load the PDS Data Set Catalog. (DATA_SET_MAP_PROJECTION and SOFTWARE are exceptions. They are not used currently to load data into the catalog.)

Templates are also used as documentation on PDS archived data sets. PDS requires that either the full set of templates be present in the CATALOG subdirectory or the file VOLINFO.TXT be present in the DOCUMENT subdirectory of an archive volume. See the File Specification and Naming chapter of this document for pointer and file names used with catalog object templates.

Depending on the type of data you are submitting, you may not need to complete every template. Your PDS Central Node Data Engineer will supply you with blank catalog templates to be completed.

Definitions and examples are provided here for your convenience. Additional examples may be obtained by contacting your Data Engineer.

The examples reflect the format to ingest metadata into the PDS catalog. Text descriptions (e.g., DATA_SET_DESC, INSTRUMENT_DESC) should not exceed 80 characters inclusive of <CR><LF> line delimiters. Of note is the underlining convention for headings and subheadings in longer text fields. Main headings are double-underlined through the use of the equal-sign key (=) which corresponds to ASCII decimal 61. Subheadings are single-underlined through the use of the hyphen key (-) which corresponds to ASCII decimal 45. This underlining convention enhances legibility, and in the future will facilitate the creation of hypertext links.

Also, PDS has adopted a convention for indenting primary headings, secondary headings, and textual descriptions to facilitate readability and to make a better presentation on the web. Primary headings start at Column 3. Text under primary headings and secondary headings start at Column 5. Text under secondary headings start at Column 7.

Again for ease of readability, there should be 2 blank lines before the start of a primary or secondary heading. If a secondary heading immediately follows a primary heading, then only 1 blank should separate the secondary heading from the primary heading.

PDS has developed a Windows based program (FORMAT70) that will automatically format the description fields of any catalog template.

```
DATA_SET_DESC                  = "

  Primary Heading - starts at Column 3
  ========================
    Text under headings start at Column 5
    more text ...
(blank line)
(blank line)
    Secondary Heading - starts at Column 5
    ---------------------------------------
      Text under subheadings start at Column 7
      more text
(blank line)
(blank line)
  Primary Heading - starts at Column 3
  ========================
(blank line)
    Secondary Heading - starts at Column 5
    ---------------------------------------
      Text under subheadings start at Column 7
      more text
```

TABLE OF CONTENTS

B.1 DATA SET

The DATA SET catalog object is used to submit information about a data set to the PDS. The catalog object includes a free-form textual description of the data set and sub-objects for identifying associated targets, hosts, and references. A separate REFERENCE object will need to be completed for any new references not already part of the PDS catalog.

(1) The DATA SET INFORMATION catalog object includes two free-form textual descriptions, DATA_SET_DESC and CONFIDENCE_LEVEL_NOTE.

NOTE: The following paragraph headings and subheadings are recommended as the minimum set of headings needed to describe a data set adequately. Additional headings and sub-headings may be added as desired. Should any of the more common headings not appear within a textual description, it will be considered not applicable to the data set.

Under DATA_SET_DESC =

Data Set Overview

A high level description of the characteristics and properties of a data set.

Parameters

Describe the primary parameters (measured or derived quantities) included in the data set, also units and sampling intervals.

Processing

Describe the overall processing used to produce the data set. Include a description of the input data (and source), processing methods or software, and primary parameters or assumptions used to produce the data set.

Data

Describe in detail each data type identified in the Data Set Overview, (e.g., Ancillary Data, Image Data, Table Data, etc.).

Ancillary Data

Describe ancillary information needed in interpreting the data set. These may or may not be provided along with the data set. Include sources or references for locating ancillary data.

Coordinate System

Describe the coordinate system or frame of reference to be used for proper interpretation of the data set.

Software

Describe software for use with the data set. This may include software supplied with the data set, or software or systems that may be accessed independently to assist in visualization or analysis of the data.

Media/Format

Describe the media on which the data set is available for distribution. Include format information that may limit the use of the data set on specific hardware platforms (e.g., binary/ascii, IBM EBCDIC format).

Under CONFIDENCE_LEVEL_NOTE =

Confidence Level Overview

A high level description of the level of confidence (e.g., reliability, accuracy, or certainty) of the data.

Review

Briefly describe any review process that took place prior to release of the data set to insure the accuracy and completeness of the data and associated documentation.

Data Coverage and Quality

Describe the overall data coverage and quality. This should include information about gaps in the data (both for times or regions). Include descriptions of how missing or poor data are flagged or filled, if applicable.

Limitations

Describe limitations on the use of the data set. For example, discuss other data required to properly interpret the data, or special processing systems expected to be used to further reduce the data set for analysis. If the data set is calibrated or otherwise corrected or derived, describe any known anomalies or uncertainties in the results.

(OTHER - Data Supplier provided):

Add any other important information in additional headings as desired (e.g., Data Compression, Time-Tagging, etc.)

(2) The DATA SET TARGET catalog object is completed for each target associated with the data set. If there is more than one target, this object is repeated.

(3) The DATA SET HOST catalog object is completed for each host/instrument pair associated with the data set. If there is more than one host/instrument pair, this object is repeated.

(4) The DATA SET REFERENCE INFORMATION catalog object is completed for each reference associated with the data set (e.g., articles, papers, memoranda, published data, etc.). If there is more than one reference, this object is repeated. A separate REFERENCE template is completed to provide the proper citation for each reference.

Important references including data set description, calibration procedures, processing software documentation, review results, etc. should be included. These can be both published and internal documents or informal memoranda.

Example:

```
/* Template: Data Set Template                                          Rev: 1993-09-24 */
/*                                                                                      */
/* Note:  Complete one for each data set. Identify multiple targets associated with     */
/*        the data set by repeating the 3 lines for the DATA_SET_TARGET object.         */
/*        Identify multiple hosts associated with the data set by repeating the 4 lines */
/*        for the DATA_SET_HOST object. Identify multiple references associated         */
/*        with the data set by repeating the 3 lines of the                             */
/*        DATA_SET_REFERENCE_INFORMATION object.                                        */

/* Hierarchy:  DATA_SET                                                                 */
/*               DATA_SET_INFORMATION                                                   */
/*               DATA_SET_TARGET                                                        */
/*               DATA_SET_HOST                                                          */
/*               DATA_SET_REFERENCE_INFORMATION                                         */

CCSD3ZF0000100000001NJPL3IF0PDSX00000001

PDS_VERSION_ID                        = PDS3
LABEL_REVISION_NOTE                   = “RSIMPSON, 1998-07-01”
RECORD_TYPE                           = STREAM

OBJECT = DATA_SET
DATA_SET_ID                           = “MGN-V-RDRS-5-GVDR-V1.0”

OBJECT                                = DATA_SET_INFORMATION
DATA_SET_NAME                         = “MGN V RDRS DERIVED GLOBAL VECTOR
                                        DATA RECORD V1.0”
DATA_SET_COLLECTION_MEMBER_FLG        = “N”
DATA_OBJECT_TYPE                      = TABLE
START_TIME                            = 1990-08-01T00:00:00
STOP_TIME                             = 1993-12-31T23:59:59
DATA_SET_RELEASE_DATE                 = 1994-07-01
PRODUCER_FULL_NAME                    = “MICHAEL J. MAURER”
DETAILED_CATALOG_FLAG                 = “N”
DATA_SET_DESC                         = “

    Data Set Overview
    =================
      The Global Vector Data Record (GVDR) is a sorted collection of scattering and emission measurements from the
      Magellan Mission.The sorting is into a grid of equal area 'pixels' distributed regularly about the planet. For data acquired
      from the same pixel but in different observing geometries, there is a second level of sorting to accommodate the different
      geometrical conditions. The 'pixel' dimension is 18.225 km. The GVDR is presented in Sinusoidal Equal Area
      (equatorial), Mercator (equatorial), and Polar Stereographic (polar) projections.

      The GVDR is intended to be the most systematic and comprehensive representation of the electromagnetic properties of
      the Venus surface that can be derived from Magellan data at this resolution. It should be useful in characterizing and
      comparing distinguishable surface units.


    Parameters
    ==========
      The Magellan data set comprises three basic data types: echoes from the nadir-viewing altimeter (ALT), echoes from the
      oblique backscatter synthetic aperture radar (SAR) imaging system, and passive radio thermal emission measurements
      made using the SAR equipment. The objective in compiling the GVDR is to obtain an accurate estimate of the surface
      backscattering function (sometimes called the specific backscatter function or 'sigma-zero') for Venus from these three
```

data types and to show its variation with incidence (polar) angle, azimuthal angle, and surface location.

The ALT data set has been analyzed to yield profiles of surface elevation [FORD&PETTENGILL1992] and estimates of surface Fresnel reflectivity and estimates of meter-scale rms surface tilts by at least two independent methods [FORD&PETTENGILL1992;TYLER1992]. The 'inversion' approach of [TYLER1992] provides, in addition, an empirical estimate of the surface backscatter function at incidence angles from nadir to as much as 10 degrees from nadir in steps of 0.5 degrees.

Statistical analysis of SAR image pixels for surface regions about 20 km (across track) by 2 km (along track) provided estimates of the surface backscatter function over narrow angular ranges (1-4 degrees) between 15 and 50 degrees from normal incidence [TYLER1992]. By combining results from several orbital passes over the same region in different observing geometries, the backscatter response over the full oblique angular range (15-50) could be compiled. In fact, the number of independent observing geometries attempted with Magellan was limited, and some of these represented changes in azimuth rather than changes in incidence (or polar) angle. Nevertheless, data from many regions were collected in more than one SAR observing geometry. Histograms of pixel values and quadratic fits to the surface backscattering function over narrow ranges of incidence angle were computed by [TYLER1992].

Passive microwave emission by the surface of Venus was measured by the Magellan radar receiver between ALT and SAR bursts.These measurements have been converted to estimates of surface emissivity [PETTENGILLETAL1992]. With certain assumptions the emissivity derived from these data should be the complement of the Fresnel reflectivity derived from the ALT echo strengths.In cases where the two quantities do not add to unity, the assumptions about a simple dielectric (Fresnel) interface at the surface of Venus must be adjusted.

Processing

The processing carried out at the Massachusetts Institute of Technology (MIT) to obtain altimetry profiles and estimates of Fresnel reflectivity and rms surface tilts has been described elsewhere [FORD&PETTENGILL1992]. In brief it involves fitting pre-computed templates to measured echo profiles; the topographic profiles, Fresnel reflectivities, and rms surface tilts are chosen to minimize differences between the data and templates in a least-squares sense. The estimates of emissivity require calibration of the raw data values and correction for attenuation and emission by the Venus atmosphere [PETTENGILLETAL1992]. These data have been collected by orbit number on a set of compact discs [FORD1992] and into a set of global maps, also distributed on compact disc [FORD1993].

At Stanford ALT-EDR tapes were the input for calculation of near-nadir empirical backscattering functions. For oblique backscatter, C-BIDR tapes from the Magellan Project and F-BIDR files obtained via Internet from Washington University were the input products. Output was collected on an orbit-by-orbit basis into a product known as the Surface Characteristics Vector Data Record (SCVDR). The SCVDR has been delivered to the Magellan Project for orbits through 2599; processing of data beginning with orbit 2600 and continuing through the end-of-Mission is spending completion of the first version of the GVDR.

Data

The GVDR data set comprises several 'tables' of results based on analysis of each of the data types described above. These include:

(1) Image Data Table
(2) Radiometry Data Table
(3) MIT ALT Data Table
(4) Stanford ALT Data Table

(1) Image Data Table
This table contains results from analysis of SAR image strips.The results are parameterized by the azimuth angle, the incidence (polar) angle, and the polarization angle. Quantities include the number of image frame lets used to compute the scattering parameters; the median, the mode, and the one-standard-deviation limits of the pixel histogram; and the three coefficients and the reference angle of the quadratic approximation to sigma-zero as a function of incidence angle.

(2) Radiometry Data Table

This table contains results from MIT analysis of the radiometry data. The results are parameterized by the azimuth angle, the incidence angle, and the polarization angle. The results include the number of radiometry footprints used to compute the estimate of thermal emissivity, the emissivity, and its variance.

(3) MIT ALT Data Table
This table contains results derived from the MIT altimetry data analysis. The results include the number of ARCDR ADF footprints used in computing the estimates of scattering properties for the pixel and estimates (and variances) of radius, rms surface tilt, and Fresnel reflectivity from the ARCDR.

(4) Stanford ALT Data Table
This table contains results from the Stanford analysis of altimetry data. Results include the number of SCVDR footprints used in computing the estimates of surface properties for this pixel, the centroid of the Doppler spectrum, the derived scattering function and the angles over which it is valid, variance of the individual points in the derived scattering function, and results of fitting analytic functions to the derived scattering function.

Ancillary Data

Ancillary data for most processing at both MIT and Stanford was obtained from the data tapes and files received from the Magellan Project. These included trajectory and pointing information for the spacecraft, clock conversion tables, spacecraft engineering data, and SAR processing parameters. For calibration of the radar instrument itself, Magellan Project reports (including some received from Hughes Aircraft Co.[BARRY1987; CUEVAS1989; SE011]) were used. Documentation on handling of data at the Jet Propulsion Laboratory was also used [BRILL&MEISL1990; SCIEDR; SDPS101].

Coordinate System

The data are presented in gridded formats, tiled to ensure that closely spaced points on the surface occupy nearby storage locations on the data storage medium. Four separate projections are used: sinusoidal equal area and Mercator for points within 89 degrees of the equator, and polar stereographic for points near the north and south poles. The projections are described by [SNYDER1987]; IAU conventions described by [DAVIESETAL1989] and Magellan Project assumptions [LYONS1988] have been adopted.

Software

A special library and several example programs are provided in source code form for reading the GVDR data files. The general-purpose example program will serve the needs of the casual user by accessing a given GVDR quantity over a specified region of GVDR pixels. More advanced users may want to write their own programs that use the GVDR library as a toolkit. The library, written in ANSI C, provides concise access methods for reading every quantity stored in the GVDR. It conveniently handles allgeometric and tiling transformations and converts any compressed qualitites to a standard native format. The general purpose program mentioned above provides an example of how to use this library.

Media/Format

The GVDR will be delivered to the Magellan Project (or its successor) using compact disc write once (CD-WO) media. Formats will be based on standards for such products established by the Planetary Data System (PDS) [PDSSR1992]."

CONFIDENCE_LEVEL_NOTE = "

Confidence Level Overview

The GVDR is intended to be the most systematic and comprehensive representation of the electromagnetic properties of the Venus surface that can be derived from Magellan data at this resolution. Nevertheless, there are limitations to what can be done with the data.

Review

The GVDR will be reviewed internally by the Magellan Project prior to release to the planetary community. The GVDR will also be reviewed by PDS.

Data Coverage and Quality

Because the orbit of Magellan was elliptical during most of its mapping operations, parts of the orbital coverage have higher resolution and higher signal-to-noise than others.

Cycle 1 Mapping

During Mapping Cycle 1, periapsis was near 10 degrees N latitude at altitudes of approximately 300 km over the surface. The altitude near the poles, on the other hand, was on the order of 3000 km. For all data types this means lower confidence in the results obtained at the poles than near the equator.

Further, the spacecraft attitude was adjusted so that the SAR antenna was pointed at about 45 degrees from nadir near periapsis; this was reduced to near 15 degrees at the poles.The objective was to compensate somewhat for the changing elevation and to provide scattering at higher incidence angles when the echo signal was expected to be strongest. The ALT antenna, at a constant 25 degree offset from the SAR antenna, followed in tandem but at angles which were not optimized for obtaining the best altimetry echo.

During Mapping Cycle 1 almost half the orbits provided SAR images of the north pole; because of the orbit inclination, ALT data never extended beyond about 85N latitude in the north and 85S in the south. No SAR images of the south pole were acquired during Mapping Cycle 1 because the SAR antenna was always pointed to the left of the ground track; the Cycle 1SAR image strip near the south pole was at a latitude equator ward of 85S.

Cycle 2 Mapping

During much of Mapping Cycle 2, the spacecraft was flown ‘backwards’ so as to provide SAR images of the same terrain but with ‘opposite side’ illumination. This adjustment also meant that the SAR could image near the Venus south pole (but not near the north pole). The ALT data continued to be limited to latitudes equator ward of 85N and 85S.

Cycle 3 Mapping

During Mapping Cycle 3 the emphasis was on obtaining SAR data from the same side as in Cycle 1 but at different incidence angles (for radar stereo). In fact, most data were acquired at an incidence angle of about 25 degrees, which meant that the ALT antenna was usually aimed directly at nadir instead of drifting from side to side, as had been the case in Cycle 1.These Cycle 3 data, therefore, may be among the best from the altimeter. Dynamic range in SAR data was larger than in Cycle1 because the incidence angle was fixed rather than varying to compensate for the changing spacecraft height.

All Cycles

It is important to remember that, since the SAR and ALT antennas were aimed at different parts of the planet during each orbit, building up a collection of composite scattering data for any single surface region requires that results from several orbits be integrated. In the case of data from polar regions, where only the SAR was able to probe, there will be no ALT data. When scheduling or other factors interrupted the systematic collection of data, there may be ALT data for some regions but no comparable SAR or radiometry data (or viceversa).

Note that for all Cycles outages played an important role in determining coverage. For example, although a goal of Cycle 3 radar mapping was radar stereo, early orbits were used to collect data at nominal incidence angles that had been missed during Cycle 1 because of thermal problems with the spacecraft. A transmitter failure during Cycle 3 caused a loss of further data. It is not within the scope of this description to provide detailed information on data coverage.

Limitations

Both the template fitting approach and the inversion approach will have their limitations in estimating overall surface properties for a region on Venus. The template calculation assumes that scattering is well-behaved at all incidence angles from 0 to 90 degrees and that a template representing that behavior can be constructed. The Hagfors function [HAGFORS1964]used by MIT, however, fails to give a finite rms surface tilt if used over this range of angles, so approximations based on a change in the scattering mechanism must be applied[HAGFORS&EVANS1968]. The inversion method [TYLER1992] is susceptible to noise at the higher incidence angles and this will corrupt solutions if not handled properly. Users of this data set should be aware that radar echoes are statistically variable and that each result has an uncertainty.

A nominal nadir footprint can be assigned to altimetry results, but this footprint is biased near periapsis because the ALT antenna is rotated about 20 degrees from nadir (during Cycle 1).Over polar regions in Cycle 1, the ALT antenna is rotated

```
          about 10 degrees to the opposite side of nadir. A more important consideration in polar regions is that the area illuminated
          by the ALT antenna is approximately 100 times as large as near periapsis because of the higher spacecraft altitude. The
          region contributing to echoes in polar regions -- and therefore the region over which estimates of Fresnel reflectivity and
          rms surface tilts apply -- is much larger than at periapsis. "

END_OBJECT                                = DATA_SET_INFORMATION

OBJECT                                    = DATA_SET_TARGET
TARGET_NAME                               = VENUS
END_OBJECT                                = DATA_SET_TARGET

OBJECT                                    = DATA_SET_HOST
INSTRUMENT_HOST_ID                        = MGN
INSTRUMENT_ID                             = RDRS
END_OBJECT                                = DATA_SET_HOST

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "BARRY1987"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "BRILL&MEISL1990"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "CUEVAS1989"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "DAVIESETAL1989"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "FORD1992"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION


OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "FORD1993"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "FORD&PETTENGILL1992"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "HAGFORS1964"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "HAGFORS&EVANS1968"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "LYONS1988"
END_OBJECT                                = DATA_SET_REFERENCE_INFORMATION

OBJECT                                    = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                          = "PDSSR1992"
```

```
END_OBJECT                          = DATA_SET_REFERENCE_INFORMATION

OBJECT                              = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                    = “PETTENGILLETAL1992”
END_OBJECT                          = DATA_SET_REFERENCE_INFORMATION

OBJECT                              = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                    = “SCIEDR”
END_OBJECT                          = DATA_SET_REFERENCE_INFORMATION

OBJECT                              = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                    = “SDPS101”
END_OBJECT                          = DATA_SET_REFERENCE_INFORMATION

OBJECT                              = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                    = “SE011”
END_OBJECT                          = DATA_SET_REFERENCE_INFORMATION

OBJECT                              = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                    = “SNYDER1987”
END_OBJECT                          = DATA_SET_REFERENCE_INFORMATION

OBJECT                              = DATA_SET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                    = “TYLER1992”
END_OBJECT                          = DATA_SET_REFERENCE_INFORMATION

END_OBJECT                          = DATA_SET

END
```

B.2 DATA SET COLLECTION

The DATA SET COLLECTION catalog object is used to link several data sets as a collection to be used and distributed together.

(1) The DATA SET COLLECTION INFO catalog object provides a description and usage, as well as other information specific to the data set collection. This object includes a free-form textual description, DATA_SET_COLLECTION_DESC.

NOTE: The paragraph headings and subheadings are recommended as the minimum set of headings needed to describe a data set collection adequately. Additional headings and sub-headings may be added as desired. Should any of the more common headings not appear within a textual description, it will be considered not applicable to the data set collection.

Under DATA_SET_COLLECTION_INFO =

Data Set Collection Overview

A high-level description of the characteristics and properties of a data set collection.

Data Set Collection Usage Overview

A high-level description of the intended use of a data set collection.

(2) The DATA SET COLL ASSOC DATA SET catalog object is repeated for each data set associated with the collection. For example, if there are three distinct data sets which make up a collection, this object will be repeated three different times, one object per data set.

(3) The DATA SET COLL REF INFO catalog object associated a reference with the data set collection. It is repeated for each reference to be identified for the collection. A separate REFERENCE template is completed to provide the associated reference citation for each new reference submitted to PDS.

Example:

```
/* Template: Data Set Collection Template                            Rev: 1993-09-24 */

/* Note:   Complete one template for each data set collection. Identify             */
/*         individual data sets that are included in the collection by              */
/*         repeating the 3 lines for the DATA_SET_COLL_ASSOC_DATA_SETS               */
/*         object. Identify each data set collection reference by                    */
/*         repeating the 3 lines for the DATA_SET_COLL_REF_INFO object.              */
/*         Also complete a separate REFERENCE template for each new                  */
/*         reference submitted to PDS.                                               */

/* Hierarchy:   DATA_SET_COLLECTION                                                  */
/*                DATA_SET_COLLECTION_INFO                                           */
/*                DATA_SET_COLL_ASSOC_DATA_SETS                                      */
/*                DATA_SET_COLLECTION_REF_INFO                                       */
```

```
OBJECT                                  = DATA_SET_COLLECTION
DATA_SET_COLLECTION_ID                  = "PREMGN-E/L/H/M/V-4/5-RAD/GRAV-V1.0"

OBJECT                                  = DATA_SET_COLLECTION_INFO
DATA_SET_COLLECTION_NAME                = "PRE-MGN E/L/H/M/V 4/5 RADAR/GRAVITY DATA V1.0"
DATA_SETS                               = 15
START_TIME                              = 1968-11-09T00:00:00
STOP_TIME                               = 1988-07-27T00:00:00
DATA_SET_COLLECTION_RELEASE_DT          = 1990-06-15
PRODUCER_FULL_NAME                      = "RAYMOND E. ARVIDSON"
DATA_SET_COLLECTION_DESC                = "

    Data Set Collection Overview
    ============================
      This entity is a collection of selected Earth-based radar data of Venus, the Moon, Mercury, and Mars, Pioneer Venus radar
      data, airborne radar images of Earth, and line of sight acceleration data derived from tracking the Pioneer Venus Orbiter
      and Viking Orbiter 2. Included are 12.6 centimeter wavelength Arecibo Venus radar images, 12.6 to 12.9cm Goldstone
      Venus radar images and altimetry data, together with altimetry, brightness temperature, Fresnel reflectivity and rms slopes
      derived from the Pioneer Venus Radar Mapper. For the Moon, Haystack 3.8 centimeter radar images and Arecibo 12.6
      and70 centimeter radar images are included. Mars data include Goldstone altimetry data acquired between 1971 and 1982
      and araster data set containing radar units that model Goldstone and Arecibo backscatter observations. Mercury data
      consist of Goldstone altimetry files. The terrestrial data were acquired over the Pisgah lava flows and the Kelso dune field
      in the Mojave Desert, California, and consist of multiple frequency, multiple incidence angle views of the same regions.
      Data set documentation is provided, with references that allow the reader to reconstruct processing histories. The entire
      data set collection and documentation are available on a CD-ROM entitled Pre-Magellan Radar and Gravity Data."

DATA_SET_COLLECTION_USAGE_DESC =        "

    Data Set Collection Usage Overview
    ==================================
      The intent of the data set collection is to provide the planetary science community with radar and gravity data similar to
      the kinds of data that Magellan will begin collecting in the summer of 1990. The data set collection will be used for pre-
      Magellan analyses of Venus and for comparisons to actual Magellan data. The entire data set collection and
      documentation are available on a CD-ROM entitled Pre-Magellan Radar and Gravity Data. A list of the hardware and
      software that may be used to read this CD-ROM can be obtained from the PDS Geosciences Discipline Node."

END_OBJECT                               = DATA_SET_COLLECTION_INFO

OBJECT                                  = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                             = "NDC8-E-ASAR-4-RADAR-V1.0"
END_OBJECT                              = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                                  = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                             = "ARCB-L-RTLS-5-12.6CM-V1.0"
END_OBJECT                              = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                                  = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                             = "ARCB-L-RTLS-4-70CM-V1.0"
END_OBJECT                              = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                                  = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                             = "ARCB-V-RTLS-4-12.6CM-V1.0"
END_OBJECT                              = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                                  = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                             = "ARCB-L-RTLS-3-70CM-V1.0"
END_OBJECT                              = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                                  = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                             = "GSSR-M-RTLS-5-ALT-V1.0"
END_OBJECT                              = DATA_SET_COLL_ASSOC_DATA_SETS
```

```
OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “GSSR-H-RTLS-4-ALT-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “GSSR-V-RTLS-5-12.6-9CM-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “HSTK-L-RTLS-4-3.8CM-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “ARCB/GSSR-M-RTLS-5-MODEL-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “P12-V-RSS-4-LOS-GRAVITY-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “P12-V-ORAD-4-ALT/RAD-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “P12-V-ORAD-5-RADAR-IMAGE-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “P12-V-ORAD-5-BACKSCATTER-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLL_ASSOC_DATA_SETS
DATA_SET_ID                     = “VO2-M-RSS-4-LOS-GRAVITY-V1.0”
END_OBJECT                      = DATA_SET_COLL_ASSOC_DATA_SETS

OBJECT                          = DATA_SET_COLLECTION_REF_INFO
REFERENCE_KEY_ID                = ARVIDSONETAL1990A
END_OBJECT                      = DATA_SET_COLLECTION_REF_INFO

END_OBJECT                       = DATA_SET_COLLECTION
END
```

B.3 DATA SET MAP PROJECTION

The DATA SET MAP PROJECTION object is one of two distinct objects that define the map projection used in creating the digital images in a PDS data set. The other associated object that completes the definition is the IMAGE MAP PROJECTION, which is fully described in Appendix A of this document.

The map projection information resides in these two objects essentially to reduce data redundancy and at the same time allow the inclusion of elements needed to process the data at the image level. Static information that is applicable to the complete data set reside in the DATA_SET_MAP_PROJECTION object while dynamic information that is applicable to the individual images reside in the IMAGE_MAP_PROJECTION object.

(1) The DATA_SET_MAP_PROJECTION catalog object unambiguously defines map projection of an image data set.

Under MAP_PROJECTION_DESC =

Map Projection Overview

A description of the map projection of the data set, indicating mathematical expressions used for latitude/longitude or line/sample transformations, line and sample projection offsets, center longitudes, etc., as well as any assumptions made in processing. (These categories of description may be subheadings indicated by single-underlining.)

Under ROTATIONAL_ELEMENT_DESCRIPTION_DESC =

Rotational Element Overview

A description of the standard used for the definition of a planet's pole orientation and prime meridian, right ascension and declination, spin angle, etc. (Please see the Planetary Science Data Dictionary for complete description.).

NOTE: The value in this field may also be a bibliographic citation to a published work containing the rotation element description. In this case there would be no need to have the 'Overview' heading. Please see the example provided below.

(2) The REFERENCE object provides citations of papers, articles, and other published and unpublished works pertinent to the data set map projection.

Example

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001

PDS_VERSION_ID                           = PDS3
LABEL_REVISION_NOTE                      = “RSIMPSON, 1998-07-01”
RECORD_TYPE                              = FIXED_LENGTH
RECORD_BYTES                             = 80

SPACECRAFT_NAME                          = MAGELLAN
TARGET_NAME                              = VENUS

OBJECT                                   = DATA_SET_MAP_PROJECTION
DATA_SET_ID                              = "MGN-V-RDRS-5-DIM-V1.0"

OBJECT                                   = DATA_SET_MAP_PROJECTION_INFO
MAP_PROJECTION_TYPE                      = "SINUSOIDAL"
MAP_PROJECTION_DESC                      = "
```

Map Projection Overview

The FMAP (Magellan Full Resolution Radar Mosaic) is presented in a Sinusoidal Equal-Area map projection. In this projection, parallels of latitude are straight lines, with constant distances between equal latitude intervals. Lines of constant longitude on either side of the projection meridian are curved since longitude intervals decrease with the cosine of latitude to account for their convergence toward the poles. This projection offers a number of advantages for storing and managing global digital data; in particular, it is computationally simple, and data are stored in a compact form.

The Sinusoidal Equal-Area projection is characterized by a projection longitude, which is the center meridian of the projection, and a scale, which is given in units of pixels/degree. The center latitude for all FMAP's is the equator. Each FMAP contains its own central meridian. The tiles that make up an FMAP all have the same central meridian as the FMAP.

Lat/Lon, Line/Sample Transformations

The transformation from latitude and longitude to line and sample is given by the following equations:

line = INT(LINE_PROJECTION_OFFSET - lat*MAP_RESOLUTION + 1.0)

sample = INT(SAMPLE_PROJECTION_OFFSET - (lon - CENTER_LONGITUDE)*MAP_RESOLUTION*cos(lat) + 1.0)

Note that integral values of line and sample correspond to center of a pixel. Lat and lon are the latitude and longitude of a given spot on the surface.

Line Projection Offset

LINE_PROJECTION_OFFSET is the line number minus one on which the map projection origin occurs. The map projection origin is the intersection of the equator and the projection longitude. The value of LINE_PROJECTION_OFFSET is positive for images starting north of the equator and is negative for images starting south of the equator.

Sample Projection Offset

SAMPLE_PROJECTION_OFFSET is the nearest sample number to the left of the projection longitude. The value of SAMPLE_PROJECTION_OFFSET is positive for images starting to the west of the projection longitude and is negative for images starting to the east of the projection longitude.

```
            Center Longitude
              CENTER_LONGITUDE is the value of the projection longitude,which is the longitude that passes through the
              center of the projection.

               The values for FMAP products will be 1408, 235, and 35.

               There are four PDS parameters that specify the latitude and longitude boundaries of an image.
               MAXIMUM_LATITUDE and MINIMUM_LATITUDE specify the latitude boundaries of the image,and
               EASTERNMOST_LONGITUDE and WESTERNMOST_LONGITUDE specify the longitudinal boundaries of
               the map.

               Definitions of other mapping parameters can be found in the Planetary Science Data Dictionary."

ROTATIONAL_ELEMENT_DESC                  = "See DAVIESETAL1989."

OBJECT                                   = DS_MAP_PROJECTION_REF_INFO
REFERENCE_KEY_ID                         = "DAVIESETAL1989"
END_OBJECT                               = DS_MAP_PROJECTION_REF_INFO

OBJECT                                   = DS_MAP_PROJECTION_REF_INFO
REFERENCE_KEY_ID                         = "BATSON1987"
END_OBJECT                               = DS_MAP_PROJECTION_REF_INFO

OBJECT                                   = DS_MAP_PROJECTION_REF_INFO
REFERENCE_KEY_ID                         = "EDWARDS1987"
END_OBJECT                               = DS_MAP_PROJECTION_REF_INFO

OBJECT                                   = DS_MAP_PROJECTION_REF_INFO
REFERENCE_KEY_ID                         = "SNYDER&JOHN1987"
END_OBJECT                               = DS_MAP_PROJECTION_REF_INFO

END_OBJECT                               = DATA_SET_MAP_PROJECTION_INFO
END_OBJECT                               = DATA_SET_MAP_PROJECTION

END
```

B.4 INSTRUMENT

The INSTRUMENT catalog object is used to submit information about an instrument to PDS. Instruments are typically associated with a particular spacecraft or earth based host, so the INSTRUMENT_HOST_ID keyword may identify either a valid SPACECRAFT_ID or EARTH_BASE_ID. The catalog object includes a textual description of the instrument and a sub-object for identifying reference information. A separate REFERENCE object will need to be completed for any new references not already part of the PDS catalog.

(1) The INSTRUMENT INFORMATION catalog object provides a description of the instrument. The following paragraph headings and suggested contents are strongly recommended as the minimal set of information necessary to adequately describe an instrument. Additional headings may be appropriate for specific instruments and these also may be added here. Should any of the recommended headings not appear within a textual description, they will be considered not applicable to the data set.

Instrument Overview

A high-level description of the characteristics and properties of an instrument.

Scientific Objectives

The scientific objectives of data obtained from this instrument.

Calibration

Methods/procedures/schedules of instrument calibration. Calibration stability, parameters, etc.

Operational Considerations

Special circumstances or events that affect the instrument's ability to acquire high quality data (which are reflected in the archive product). Examples might be spacecraft charging, thruster firings, contamination from other instruments, air quality, temperatures, etc.

Detectors

General description of detector(s). Type of detector used. Sensitivity and noise levels. Detector fields of view, geometric factors, etc. Instrument/detector mounting descriptions (offset angles, pointing positions, etc.)

Electronics

Description of the instrument electronics and internal data processing (A-D converter).

Filters

Description of instrument filters and filter calibrations (filter type, center wavelength, min/max wavelength) if applicable.

Optics

Description of instrument optics (focal lengths, transmittance, diameter, resolution, t_number, etc.) if applicable.

Location

Latitude and longitude location, for earth based instruments.

Operational Modes

Description of instrument configurations for data acquisitions. Description of “modes” (scan, gain, etc.) of data acquisition and of measured parameter(s) and/or data sampling rates or schemes used in each mode.

Subsystems

Logical subsystems of the instrument. Description of each subsystem, how it's used, which “modes” make use of which subsystem, etc.

Measured Parameters

Description of what the instrument directly measures (particle counts, magnetic field components, radiance, current/voltage ratios, etc.) Description and definition of these measurements (min/max, noise levels, units, time interval between measurements, etc.)

(OTHER - Data Supplier provided):

Any other important information in additional headings as desired (e.g. Data Reduction, Data Compression, Time-Tagging, Diagnostics, etc.)

(2) The INSTRUMENT REFERENCE INFO catalog object associates a reference with the instrument description. It is repeated for each reference identified for the instrument. A separate REFERENCE template is completed to provide the associated reference citation for each reference.

Include any important references such as instrument description and calibration documents. These can be both published and internal documents or informal memoranda.

Example:

```
/* Template: Instrument Template                    Rev: 1993-09-24                   */

/* Note:   Complete one template for each instrument. Identify each                     */
/*         instrument reference by repeating the 3 lines for the                        */
/*         INSTRUMENT_REFERENCE_INFO object. Also complete a separate                   */
/*         REFERENCE template for each new reference submitted to PDS.                  */

/* Hierarchy:   INSTRUMENT                                                              */
/*                INSTRUMENT_INFORMATION                                                */
/*                INSTRUMENT_REFERENCE_INFO                                             */

CCSD3ZF0000100000001NJPL3IF0PDSX00000001
```

```
PDS_VERSION_ID                              = PDS3
LABEL_REVISION_NOTE                         = “RSIMPSON, 1998-07-01”
RECORD_TYPE                                 = STREAM

OBJECT = INSTRUMENT
INSTRUMENT_HOST_ID                          = “MGN”
INSTRUMENT_ID                               = “RDRS”

OBJECT                                      = INSTRUMENT_INFORMATION
INSTRUMENT_NAME                             = “RADAR SYSTEM”
INSTRUMENT_TYPE                             = “RADAR”
INSTRUMENT_DESC                             = “
```

Instrument Overview

The Magellan radar system included a 3.7 m diameter high gain antenna (HGA) for SAR and radiometry and a smaller fan-beam antenna (ALTA) for altimetry. The system operated at 12.6 cm wavelength. Common electronics were used in SAR, altimetry, and radiometry modes. The SAR operated in a burst mode; altimetry and radiometry observations were interleaved with the SAR bursts.

Radiometry data were obtained by spending a portion of the time between SAR bursts and after altimeter operation in a passive (receive-only) mode, with the HGA antenna capturing the microwave thermal emission from the planet. Noise power within the 10-MHz receiver bandwidth was detected and accumulated for50 ms. To reduce the sensitivity to receiver gain changes in this mode, the receiver was connected on alternate bursts first to a comparison dummy load at a known physical temperature and then to the HGA. The short-term temperature resolution was about 2 K; the long-term absolute accuracy after calibration was about 20 K.

The radar was manufactured by Hughes Aircraft Company and the 'build date' is taken to be 1989-01-01. The radar dimensions were 0.304 by 1.35 by 0.902 (height by length by width in meters) and the mass was 126.1 kg.

Instrument Id	: RDRS
Instrument Host Id	: MGN
Pi PDS User Id	: GPETTENGILL
Instrument Name	: RADAR SYSTEM
Instrument Type	: RADAR
Build Date	: 1989-01-01
Instrument Mass	: 126.100000
Instrument Length	: 1.350000
Instrument Width	: 0.902000
Instrument Height	: 0.304000
Instrument Manufacturer Name	: HUGHES AIRCRAFT

Platform Mounting Descriptions

The spacecraft +Z axis vector was in the nominal direction of the HGA boresight. The +X axis vector was parallel to the nominal rotation axis of the solar panels. The +Y axis vector formed a right-handed coordinate system and was in the nominal direction of the star scanner boresight. The spacecraft velocity vector was in approximately the -Y direction when the spacecraft was oriented for left-looking SAR operation. The nominal HGA polarization was linear in the y-direction.

Cone Offset Angle:	0.00
Cross Cone Offset Angle:	0.00
Twist Offset Angle:	0.00

The altimetry antenna boresight was in the x-z plane 25 degrees from the +Z direction and 65 degrees from the +X direction. The altimetry antenna was aimed approximately toward nadir during nominal radar operation. The altimetry antenna polarization was linear in the y-direction.

The medium gain antenna boresight was 70 degrees from the +Z direction and 20 degrees from the -Y direction. The low gain antenna was mounted on the back of the HGA feed; it's boresight was in the +Z direction and it had a hemispherical radiation pattern.

Principal Investigator

The Principal Investigator for the radar instrument was Gordon H. Pettengill.

For more information on the radar system see the papers by [JOHNSON1990] and [SAUNDERSETAL1990].

Scientific Objectives

See MISSION_OBJECTIVES_SUMMARY under MISSION.

Operational Considerations

The Magellan radar system was used to acquire radar back-scatter(SAR) images, altimetry, and radiometry when the spacecraft was close to the planet. Nominal operation extended from about 20minutes before periapsis until about 20 minutes after periapsis.In the SAR mode output from the radar receiver was sampled, blocks of samples were quantized using an adaptive procedure, and the results were stored on tape. In the altimetry mode samples were recorded directly, without quantization. Radiometry measurements were stored in the radar header records. During most of the remainder of each orbit, the HGA was pointed toward Earth and the contents of the tape recorder were transmitted to a station of the DSN at approximately 270 kilobits/second. SAR, altimetry, and radiometry data were then processed using ground software into images, altimetry profiles, estimates of backscatter coefficient, emissivity, and other quantities.

Calibration

The radar was calibrated before flight using an active electronic target simulator [CUEVAS1989].

Operational Modes

The Magellan radar system consisted of the following sections, each of which operated in the following modes:

Section	Mode
SAR	Synthetic Aperture Radar (SAR)
ALT	Altimetry
RAD	Radiometry

(1) SAR Characteristics

In the Synthetic Aperture Radar mode, the radar transmitted bursts of phase-modulated pulses through its high gain antenna. Echo signals were captured by the antenna, simple dat the receiver output, and stored on tape after being quantized to reduce data volume. Pulse repetition rate and incidence angle were chosen to meet a minimum signal-to-noise ratio requirement (8 dB) for image pixels after ground processing. Multiple looks were used in processing to reduce speckle noise. Incidence angles varied from about 13 degree sat the pole to about 44 degrees at periapsis during normal mapping operations (e.g., Cycle 1); but other 'look angle profiles' were used during the mission.

Peak transmit power	: 350 watts
Transmitted pulse length	: 26.5 microsecs
Pulse repetition frequency	: 4400-5800 per sec
Time bandwidth product	: 60
Inverse baud width	: 2.26 MHz
Data quantization (I and Q)	: 2 bits each
Recorded data rate	: 750 kilobits/sec
Polarization (nominal)	: linear horizontal
HGA half-power full beam width	: 2.2 deg (azimuth)
	: 2.5 deg (elev)
one-way gain (from SAR RF port)	: 35.7
dBi System temperature (viewing Venus)	: 1250 K
Surface resolution (range)	: 120-360 m
(along track)	: 120-150 m
Number of looks	: 4 or more
Swath width	: 25 km (approx)

```
               Antenna look angle                                    : 13-47 deg
               Incidence angle on surface                            : 18-50 deg

               Data Path Type                                        : RECORDED DATA
               PLAYBACK Instrument Power Consumption                 : UNK


               (2) ALT Characteristics
               After SAR bursts (typically several times a second) groups of altimeter pulses were transmitted from a dedicated
               fan beam altimeter antenna (ALTA) directed toward the spacecraft's nadir. Output from the radar receiver was
               sampled, and the samples were stored on tape for transmission to Earth. During nominal left-looking SAR operation
               the ALTA pointed approximately 20 deg to the left of the spacecraft ground track at periapsis and about 10 deg to
               the right of the ground track near the north and south pole.

               Data quantization (I and Q)                           : 4 bits each
               Recorded data rate                                    : 35 kbs
               Polarization                                          : linear
               ALTA half-power full beam width
               (along track)                                         : 11 deg
               (cross track)                                         : 31 deg
               one-way gain referenced to ALT RF port                : 18.9
               dBi ALTA offset from HGA                              : 25 deg
               Burst interval                                        : 0.5-1.0 sec
               duration                                              : 1.0 millisec
               Dynamic range                                         : 30 dB (or more)

               Data Path Type                                        : RECORDED DATA
               PLAYBACK Instrument Power Consumption                 : UNK


               (3) RAD Characteristics
               Radiometry measurements were made by the radar receiver and HGA in a receive-only mode that was activated after
               the altimetry mode to record the level of microwave radio thermale mission from the planet. Noise power within the
               10-MHz receiver bandwidth was detected and accumulated for 50 ms. To reduce the sensitivity to receiver gain
               changes in this mode, the receiver was connected on alternate bursts first to a comparison dummy load at a known
               physical temperature and then to the HGA. The short-term temperature resolution was about 2K; the long-term
               absolute accuracy after calibration was about20 K. At several times during the mission, radiometry measurements
               were carried out using known cosmic radio sources.

               Receiver Bandwidth                                    : 10 MHz
               Integration Time                                      : 50 millisecs
               Polarization (nominal)                                : linear horizontal
               Data Quantization                                     : 12 bits
               Data Rate                                             : 10-48 bits/sec
               HGA half-power full beam width                        : 2.2 deg
               System temperature (viewing Venus)                    : 1250 K
               Antenna look angle                                    : 13-47 deg
               Incidence angle on surface                            : 18-50 deg
               Surface resolution (along track)                      : 15-120 km
               (cross track)                                         : 20-125 km

               Data Path Type                     : RECORDED DATA PLAYBACK
               Instrument Power Consumption       : UNK ”

END_OBJECT                                         = INSTRUMENT_INFORMATION

OBJECT                                            = INSTRUMENT_REFERENCE_INFO
REFERENCE_KEY_ID                                  = “CUEVAS1989”
END_OBJECT                                        = INSTRUMENT_REFERENCE_INFO
```

```
OBJECT                                  = INSTRUMENT_REFERENCE_INFO
REFERENCE_KEY_ID                        = “JOHNSON1990”
END_OBJECT                              = INSTRUMENT_REFERENCE_INFO

OBJECT                                  = INSTRUMENT_REFERENCE_INFO
REFERENCE_KEY_ID                        = “SAUNDERSETAL1990”
END_OBJECT                              = INSTRUMENT_REFERENCE_INFO

END_OBJECT                              = INSTRUMENT

END
```

B.5 INSTRUMENT HOST

The INSTRUMENT HOST catalog object is used to describe a variety of instrument hosts, such as a spacecraft or an earth based observatory.

(1) The INSTRUMENT HOST INFORMATION catalog object provides a textual description that may be used to describe any important information about an instrument host. For spacecraft, this typically includes paragraphs on the various subsystems. Earthbased instrument host descriptions may focus on geographic and facility elements.

Instrument Host Overview
A high-level description of the characteristics and properties of the instrument host.

(2) The INSTRUMENT HOST REFERENCE INFO catalog object is completed for each reference associated with the host. If there is more than one reference, this object is repeated. A separate REFERENCE template is completed to provide the proper citation for each reference.

Example:

```
/* Template: Instrument Host Template                                    Rev: 1993-09-24 */

/* Note:   Complete one template for each instrument host. Identify each                 */
/*         instrument host reference by repeating the 3 lines for the                    */
/*         INSTRUMENT_HOST_REFERENCE_INFO object. Also complete a separate               */
/*         REFERENCE template for each new reference submitted to PDS.                   */

/* Hierarchy:   INSTRUMENT_HOST                                                          */
/*                INSTRUMENT_HOST_INFORMATION                                            */
/*                INSTRUMENT_HOST_REFERENCE_INFO                                         */

CCSD3ZF0000100000001NJPL3IF0PDSX00000001

PDS_VERSION_ID                          = PDS3
LABEL_REVISION_NOTE                     = “RSIMPSON, 1998-07-01”
RECORD_TYPE                             = “STREAM”

OBJECT                                  = INSTRUMENT_HOST
INSTRUMENT_HOST_ID                      = “MGN”

OBJECT                                  = INSTRUMENT_HOST_INFORMATION
INSTRUMENT_HOST_NAME                    = “MAGELLAN”
INSTRUMENT_HOST_TYPE                    = “SPACECRAFT”
INSTRUMENT_HOST_DESC                    = “

    Instrument Host Overview
      The Magellan spacecraft was built by the Martin Marietta Corporation. The spacecraft structure included four major
      sections: High-Gain Antenna (HGA), Forward Equipment Module (FEM), Spacecraft Bus (including the solar array), and
      the Orbit Insertion Stage. Spacecraft subsystems included those for thermal control, power, attitude control, propulsion,
      command data and data storage, and telecommunications.

      The Magellan telecommunications subsystem contained all the hardware necessary to maintain communications between
```

```
          Earth and the spacecraft. The subsystem contained the radio frequency subsystem, the LGA, MGA, and HGA. The RFS
          performed the functions of carrier transponding, command detection and decoding, and telemetry modulation. The
          spacecraft was capable of simultaneous X-band and S-band uplink and downlink operations. The S-band operated at a
          transmitter power of 5 W, while the X-band operated at a power of 22 W. Uplink data rates were 31.25 and 62.5 bps (bits
          per second) with downlink data rates of 40 bps (emergency only), 1200 bps (real-time engineering rate), 115.2 kbps
          (kilobits per second) (radar down link backup), and 268.8 kbps (nominal).

          For more information on the Magellan spacecraft see the papers by [SAUNDERSETAL1990] and
          [SAUNDERSETAL1992]. "

END_OBJECT                                 = INSTRUMENT_HOST_INFORMATION

OBJECT                                    = INSTRUMENT_HOST_REFERENCE_INFO
REFERENCE_KEY_ID                          = "SAUNDERSETAL1990"
END_OBJECT                                = INSTRUMENT_HOST_REFERENCE_INFO

OBJECT                                    = INSTRUMENT_HOST_REFERENCE_INFO
REFERENCE_KEY_ID                          = "SAUNDERSETAL1992"
END_OBJECT                                = INSTRUMENT_HOST_REFERENCE_INFO

END_OBJECT                                = INSTRUMENT_HOST
END
```

B.6 INVENTORY

The INVENTORY catalog object shall be completed once for each node that is responsible for orderable data sets from the PDS catalog. This object provides the inventory information necessary to facilitate the ordering of these data sets.

(1) The INVENTORY DATA SET INFO catalog object identifies a product through the product data set id. This object is repeated for each orderable and cataloged PDS data set.

(2) The INVENTORY NODE MEDIA INFO catalog object provides information about data set distribution medium. This object is repeated for each type of distribution medium.

Example:

```
/* Template: InventoryTemplate                                        Rev: 1990-03-20 */

/* Note:   The INVENTORY template shall be completed once for each node that is responsible      */
/*         for orderable data sets from the PDS catalog. The following hierarchy of templates provide      */
/*         the necessary inventory information which will facilitate the ordering of these data sets.      */

/* Hierarchy:   INVENTORY                                                  */
/*              INVENTORY_DATA_SET_INFO                                    */
/*              INVENTORY_NODE_MEDIA_INFO                                  */

OBJECT                              = INVENTORY
NODE_ID                             = “IMAGING”

OBJECT                              = INVENTORY_DATA_SET_INFO
PRODUCT_DATA_SET_ID                 = “VG2-N-ISS-2-EDR-V1.0”

OBJECT                              = INVENTORY_NODE_MEDIA_INFO
MEDIUM_TYPE                         = “MAG TAPE”
MEDIUM_DESC                         = “INDUSTRY STD 1/2IN;1600 OR 6250 BPI”
COPIES                              = 1
INVENTORY_SPECIAL_ORDER_NOTE        = “Not applicable.”
END_OBJECT                          = INVENTORY_NODE_MEDIA_INFO

OBJECT                              = INVENTORY_NODE_MEDIA_INFO
MEDIUM_TYPE                         = “CD-ROM”
MEDIUM_DESC                         = “Compact Disk”
COPIES                              = 1
INVENTORY_SPECIAL_ORDER_NOTE        = “Not applicable.”
END_OBJECT                          = INVENTORY_NODE_MEDIA_INFO

END_OBJECT                          = INVENTORY_DATA_SET_INFO
END_OBJECT                          = INVENTORY

OBJECT                              = INVENTORY
NODE_ID                             = “NSSDC”

OBJECT                              = INVENTORY_DATA_SET_INFO
PRODUCT_DATA_SET_ID                 = “VG2-N-ISS-2-EDR-V1.0”
```

```
OBJECT = INVENTORY_NODE_MEDIA_INFO
MEDIUM_TYPE                          = “CD-ROM”
MEDIUM_DESC                          = “Compact Disk”
COPIES                               = 1
INVENTORY_SPECIAL_ORDER_NOTE         = “Not applicable.”
END_OBJECT                           = INVENTORY_NODE_MEDIA_INFO

END_OBJECT                           = INVENTORY_DATA_SET_INFO
END_OBJECT                           = INVENTORY
END
```

B.7 MISSION

The MISSION catalog object is used to submit information about a mission or campaign to PDS. Sub-objects are included for identifying associated instrument hosts, targets, and references. A separate REFERENCE object will need to be completed for any new references not already a part of the PDS catalog.

(1) The MISSION INFORMATION catalog object provides start and stop times and textual descriptions, MISSION_DESC and MISSION_OBJECTIVES_SUMMARY. Suggested contents include agency involvement, spacecraft/observatory utilized, mission scenario including phases, technology and scientific objectives.

Under MISSION_DESC =

Mission Overview
A high-level description of a mission.

Mission Phases
A description of each phase of a mission, starting with the pre-launch phase and continuing through end-of-mission. This includes start and stop times of each phase, intended operations, targets, and mission phase objectives.

Under MISSION_OBJECTIVES_SUMMARY =

Mission Objectives Overview
A high-level description of the objectives of the mission.

(2) The MISSION HOST catalog object is completed for each instrument host associated with the mission or campaign. If there is more than one instrument host involved in the mission, this object is repeated.

(3) The MISSION TARGET catalog object is completed for each target associated with an instrument host. If there is more than one target for a given host, this object is repeated.

(4) The MISSION REFERENCE INFORMATION catalog object is completed for each reference associated with the mission. If there is more than one reference, this object is repeated. A separate REFERENCE template is completed to provide the proper citation for each reference.

Example:

```
/* Template: Mission Template                     Rev: 1993-09-24                         */

/* Note:  Complete one template for each mission or campaign. Identify                    */
/*        multiple hosts associated with the mission by repeating the                     */
/*        lines beginning and ending with the MISSION_HOST values. For                    */
/*        each instrument_host identified, repeat the 3 lines for the                     */
/*        MISSION_TARGET object for each target associated with the host.                 */
/*        Also complete a separate REFERENCE template for each new                        */
/*        reference submitted to PDS.                                                     */

/* Hierarchy: MISSION                                                                     */
/*        MISSION_INFORMATION                                                             */
/*        MISSION_HOST                                                                    */
/*        MISSION_TARGET                                                                  */
/*        MISSION_REFERENCE_INFORMATION                                                   */

CCSD3ZF0000100000001NJPL3IF0PDSX00000001

PDS_VERSION_ID                                    = PDS3
LABEL_REVISION_NOTE                               = “RSIMPSON, 1998-07-01”
RECORD_TYPE                                       = STREAM

OBJECT                                            = MISSION
MISSION_NAME                                      = “MAGELLAN”

OBJECT                                            = MISSION_INFORMATION
MISSION_START_DATE                                = 1989-05-04
MISSION_STOP_DATE                                 = UNK
MISSION_ALIAS_NAME                                = “Venus Radar Mapper (VRM)”
MISSION_DESC= “

      Mission Overview
      ================
        The Magellan spacecraft was launched from the Kennedy Space Center on 4 May 1989. The spacecraft was deployed
        from the Shuttle cargo bay after the Shuttle achieved parking orbit.Magellan, using an inertial upper stage rocket, was
        then placed into a Type IV transfer orbit to Venus where it carried out radar mapping and gravity studies starting in August
        1990. The Mission has been described in many papers including two special issues of the Journal of Geophysical Research
        [VRMPP1983;SAUNDERSETAL1990; JGRMGN1992]. The radar system is also described in [JOHNSON1990].

        The aerobraking phase of the mission was designed to change the Magellan orbit from eccentric to nearly circular. This
        was accomplished by dropping periapsis to less than 150 km above the surface and using atmospheric drag to reduce the
        energy in the orbit. Aerobraking ended on 3 August 1993, and periapsis was boosted above the atmosphere leaving the
        spacecraft in an orbit that was 540 km above the surface at apoapsis and 197 km above the surface at periapsis. The orbit
        period was 94 minutes. The spacecraft remained on its medium-gain antenna in this orbit until Cycle 5 began officially
        on 16 August 1993.

        During Cycles 5 and 6 the orbit was low and approximately circular. The emphasis was on collecting high-resolution
        gravity data. Two bistatic surface scattering experiments were conducted, one on 6 October (orbits 9331, 9335, and 9336)
        and the second on 9 November (orbits 9846-9848).


      Mission Phases
      ==============
        Mission phases were defined for significant spacecraft activity periods. During orbital operations a ‘cycle’ was
        approximately the time required for Venus to rotate once under the spacecraft (about 243 days). But there were orbit
        adjustments and other activities that made some mapping cycles not strictly contiguous and slightly longer or shorter than
        the rotation period.
```

PRELAUNCH

The prelaunch phase extended from delivery of the spacecraft to Kennedy Space Center until the start of the launch countdown.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1988-09-01
Mission Phase Stop Time	: 1989-05-04
Spacecraft Operations Type	: ORBITER

LAUNCH

The launch phase extended from the start of launch countdown until completion of the injection into the Earth-Venus trajectory.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1989-05-04
Mission Phase Stop Time	: 1989-05-04
Spacecraft Operations Type	: ORBITER

CRUISE

The cruise phase extended from injection into the Earth-Venus trajectory until 10 days before Venus orbit insertion.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1989-05-04
Mission Phase Stop Time	: 1990-08-01
Spacecraft Operations Type	: ORBITER

ORBIT INSERTION

The Venus orbit insertion phase extended from 10 days before Venus orbit insertion until burnout of the solid rocket injection motor.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1990-08-01
Mission Phase Stop Time	: 1990-08-10
Spacecraft Operations Type	: ORBITER

ORBIT CHECKOUT

The orbit trim and checkout phase extended from burnout of the solid rocket injection motor until the beginning of radar mapping.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1990-08-10
Mission Phase Stop Time	: 1990-09-15
Spacecraft Operations Type	: ORBITER

MAPPING CYCLE 1

The first mapping cycle extended from completion of the orbit trim and checkout phase until completion of one cycle of radar mapping (approximately 243 days).

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1990-09-15
Mission Phase Stop Time	: 1991-05-15
Spacecraft Operations Type	: ORBITER

MAPPING CYCLE 2

The second mapping cycle extended from completion of the first mapping cycle through an additional cycle of mapping.Acquisition of 'right-looking' SAR data was emphasized. Radio occultation measurements were carried out on orbits 3212-3214.A period of battery reconditioning followed completion of Cycle 2.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1991-05-16
Mission Phase Stop Time	: 1992-01-17
Spacecraft Operations Type	: ORBITER

MAPPING CYCLE 3

The third mapping cycle extended from completion of battery reconditioning through an additional cycle of mapping (approximately 243 days). Acquisition of 'stereo' SAR data was emphasized. The last orbit in the third cycle was orbit5747.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1992-01-24
Mission Phase Stop Time	: 1992-09-14
Spacecraft Operations Type	: ORBITER

MAPPING CYCLE 4

The fourth mapping cycle extended from completion of the third mapping cycle through an additional cycle of mapping. Acquisition of radio tracking data for gravity studies was emphasized. Radio occultation measurements were carried out on orbits 6369, 6370, 6471, and 6472. Because of poor observing geometry for gravity data collection at the beginning of the cycle, this cycle was extended 10 days beyond the nominal 243 days. Orbits included within the fourth cycle were 5748 through 7626. Periapsis was lowered on orbit 5752to improve sensitivity to gravity features in Cycle 4.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1992-09-14
Mission Phase Stop Time	: 1993-05-25
Spacecraft Operations Type	: ORBITER

AEROBRAKING

The aerobraking phase extended from completion of the fourth mapping cycle through achievement of a near-circular orbit.Circularization was achieved more quickly than expected; the first gravity data collection in the circular orbit was not scheduled until 11 days later. Orbits included within the aerobraking phase were 7627 through 8392.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1993-05-26
Mission Phase Stop Time	: 1993-08-05
Spacecraft Operations Type	: ORBITER

MAPPING CYCLE 5

The fifth mapping cycle extended from completion of the aerobraking phase through an additional cycle of mapping (approximately 243 days). Acquisition of radio tracking data for gravity studies was emphasized. The first orbit in the fifth cycle was orbit 8393.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1993-08-16
Mission Phase Stop Time	: 1994-04-15
Spacecraft Operations Type	: ORBITER

MAPPING CYCLE 6

The sixth mapping cycle extended from completion of the fifth mapping cycle through an additional cycle of mapping (approximately 243 days). Acquisition of radio tracking data for gravity studies was emphasized. The first orbit in the sixth cycle was orbit 12249.

Spacecraft Id	: MGN
Target Name	: VENUS
Mission Phase Start Time	: 1994-04-16
Mission Phase Stop Time	: TBD
Spacecraft Operations Type	: ORBITER”

MISSION_OBJECTIVES_SUMMARY = “

Mission Objectives Overview

Volcanic and Tectonic Processes

Magellan images of the Venus surface show widespread evidence for volcanic activity. A major goal of the Magellan mission was to provide a detailed global characterization of volcanic land forms on Venus and an understanding of the mechanics of volcanism in the Venus context. Of particular interest was the role of volcanism in transporting heat through the lithosphere.While this goal will largely be accomplished by a careful analysis of images of volcanic features and of the geological relationships of these features to tectonic and impact structures, an essential aspect of characterization will be an integration of image data with altimetry and other measurements of surface properties....

For more information on volcanic and tectonic investigations see papers by [HEADETAL1992] and [SOLOMONETAL1992], respectively.

Impact Processes

The final physical form of an impact crater has meaning only when the effects of the cratering event and any subsequent modification of the crater can be distinguished. To this end, a careful search of the SAR images can identify and characterize both relatively pristine and degraded impact craters, together with their ejecta deposits (in each size range) as well as distinguishing impact craters from those of volcanic origin.The topographic measures of depth-to-diameter ratio, ejecta thickness distribution as a function of distance from the crater, and the relief of central peaks contribute to this documentation.

.

For more information on investigations of impact processes see[SCHABERETAL1992].

Erosional, Depositional, and Chemical Processes

The nature of erosional and depositional processes on Venus is poorly known, primarily because the diagnostic landforms typically occur at a scale too small to have been resolved in Earth-based or Venera 15/16 radar images. Magellan images show wind eroded terrains, landforms produced by deposition (dunefields), possible landslides and other down slope movements, as well as aeolian features such as radar bright or dark streaks 'downwind' from prominent topographic anomalies. One measure of weathering, erosion, and deposition is provided by the extent to which soil covers the surface (for Venus, the term soil is used for porous material, as implied by its relatively low value of bulk dielectric constant). The existence of such material, and its dependence on elevation and geologic setting, provide important insights into the interactions that have taken place between the atmosphere and the lithosphere.

```
                .
                For more information on erosional, depositional, and chemical processes see papers by [ARVIDSONETAL1992],
                [GREELEYETAL1992],and [GREELEYETAL1994].


            Isostatic and Convective Processes
                Topography and gravity are intimately and inextricably related, and must be jointly examined when undertaking
                geophysical investigations of the interior of a planet, where isostatic and convective processes dominate.
                Topography provides a surface boundary condition for modeling the interior density of Venus.

                For more information on topography and gravity see papers by[FORD&PETTENGILL1992],
                [KONOPLIVETAL1993], and[MCNAMEEETAL1993]. ”

END_OBJECT                                  = MISSION_INFORMATION

OBJECT                                      = MISSION_HOST
INSTRUMENT_HOST_ID                          = “MGN”

OBJECT                                      = MISSION_TARGET
TARGET_NAME                                 = “VENUS”
END_OBJECT                                  = MISSION_TARGET
END_OBJECT                                  = MISSION_HOST

OBJECT                                      = MISSION_REFERENCE_INFORMATION
REFERENCE_KEY_ID                            = “ARVIDSON1991”
END_OBJECT                                  = MISSION_REFERENCE_INFORMATION

OBJECT                                      = MISSION_REFERENCE_INFORMATION
REFERENCE_KEY_ID                            = “ARVIDSONETAL1992”
END_OBJECT                                  = MISSION_REFERENCE_INFORMATION

OBJECT                                      = MISSION_REFERENCE_INFORMATION
REFERENCE_KEY_ID                            = “CAMPBELLETAL1992”
END_OBJECT                                  = MISSION_REFERENCE_INFORMATION
          .
          .
          .
OBJECT                                      = MISSION_REFERENCE_INFORMATION
REFERENCE_KEY_ID                            = “TYLER1992”
END_OBJECT                                  = MISSION_REFERENCE_INFORMATION

OBJECT                                      = MISSION_REFERENCE_INFORMATION
REFERENCE_KEY_ID                            = “VRMPP1983”
END_OBJECT                                  = MISSION_REFERENCE_INFORMATION

END_OBJECT                                  = MISSION
END
```

B.8 PERSONNEL

The PERSONNEL catalog object is used to provide new or updated information for personnel associated with PDS in some capacity. This includes data suppliers and producers for data sets or volumes archived with PDS, as well as PDS node personnel and contacts within other agencies and institutions.

(1) The PERSONNEL INFORMATION catalog object provides name, address, telephone, and related information.

(2) The PERSONNEL ELECTRONIC MAIL catalog object provides electronic mail information for personnel. This object may be repeated if more than one electronic mail address is applicable.

Example

```
/* Template: Personnel Template Rev: 1993-09-24 */

/* Note:   Complete one for each new PDS user, data supplier, or data                 */
/*         producer. If more than one electronic mail address is available            */
/*         repeat the lines for the PERSONNEL_ELECTRONIC_MAIL object.                 */

/* Hierarchy:   PERSONNEL                                                             */
/*                PERSONNEL_INFORMATION                                               */
/*                PERSONNEL_ELECTRONIC_MAIL                                           */

OBJECT                                   = PERSONNEL
RECORD_TYPE                              = STREAM

PDS_USER_ID                              = PFORD

OBJECT                                   = PERSONNEL_INFORMATION
FULL_NAME                                = “PETER G. FORD”
LAST_NAME                                = FORD
TELEPHONE_NUMBER                         = “6172536485”
ALTERNATE_TELEPHONE_NUMBER               = “6172534287”
FAX_NUMBER                               = “6172530861”
INSTITUTION_NAME                         = “MASSACHUSETTS INSTITUTE OF TECHNOLOGY”
NODE_ID                                  = “GEOSCIENCE”
PDS_AFFILIATION                          = “NODE OPERATIONS MANAGER”
PDS_ADDRESS_BOOK_FLAG                    = Y
REGISTRATION_DATE                        = 1990-02-06
ADDRESS_TEXT                             = “Massachusetts Institute of Technology \n
                                         Center for Space Research Building 37-601Cambridge, MA 02139"
END_OBJECT                               = PERSONNEL_INFORMATION

OBJECT                                   = PERSONNEL_ELECTRONIC_MAIL
ELECTRONIC_MAIL_ID                       = “PGF@SPACE.MIT.EDU”
ELECTRONIC_MAIL_TYPE                     = “INTERNET”
PREFERENCE_ID                            = 1
END_OBJECT                               = PERSONNEL_ELECTRONIC_MAIL

OBJECT                                   = PERSONNEL_ELECTRONIC_MAIL
ELECTRONIC_MAIL_ID                       = “PFORD”
```

```
ELECTRONIC_MAIL_TYPE                     = “NASAMAIL”
PREFERENCE_ID                            = 2
END_OBJECT                               = PERSONNEL_ELECTRONIC_MAIL

OBJECT                                   = PERSONNEL_ELECTRONIC_MAIL
ELECTRONIC_MAIL_ID                       = “JPLPDS::PFORD”
ELECTRONIC_MAIL_TYPE                     = “NSI/DECNET”
PREFERENCE_ID                            = 3
END_OBJECT                               = PERSONNEL_ELECTRONIC_MAIL

END_OBJECT                                = PERSONNEL
END
```

B.9 REFERENCE

The REFERENCE catalog object is completed for each reference associated with a mission, instrument host, instrument, data set, or data set collection catalog object. Submit any important references, including both published and unpublished internal documents or informal memoranda. This also may include references to published data, such as PDS archive volumes. A copy of an unpublished reference should be forwarded to the PDS node responsible for your data set archive, whenever possible.

(1) The REFERENCE catalog object provides a reference citation and a unique identifier for every reference associated with the PDS data archive.

Example:

```
/* Template: Reference Template                    Rev: 1993-09-24                           */

/* Note:   This template shall be completed for each reference associated with a mission,    */
/*         instrument host, instrument, data set, or data set collection template.          */

OBJECT                                             = REFERENCE
REFERENCE_KEY_ID                                   = “GURNETTETAL1991”
REFERENCE_DESC                                     = “Garnet, D.A., W.S. Kurth, A. Roux, R. Gendrin, C. F. Kennel, S. J.
Bolton, Lightning and Plasma Wave Observations from the Galileo Flyby of Venus, Science, 253, 1522, 1991."
END_OBJECT                                         = REFERENCE

OBJECT                                             = REFERENCE
REFERENCE_KEY_ID                                   = ARVIDSONETAL1990A
REFERENCE_DESC                                     = “Arvidson, R.E., E.A. Guinness, S.
Slavney, D. Acevedo, J. Hyon, and M. Martin, Pre-Magellan radar and gravity data, Jet Propulsion Laboratory, CDROM
(USA_NASA_JPL_MG_1001).”
END_OBJECT                                         = REFERENCE
END
```

B.10 SOFTWARE

The SOFTWARE catalog object is completed for each software program registered in the PDS Software Inventory. This Inventory includes software available within the Planetary Science community, including software on PDS archive volumes. Of interest are any applications, tools, or libraries that have proven useful for the display, analysis, formatting, transformation, or preparation of either science data or meta-data for the PDS archives.

(1) The SOFTWARE catalog object provides general information about the software tool including a description, availability information, and dependencies.

Example:

```
/* Template: Software Template                                        Rev: 1998-12-01    */

/* Note:   This template should be completed to register software in the                  */
/*         PDS Software Inventory.                                                        */


OBJECT                                    = SOFTWARE
  SOFTWARE_ID                             = NASAVIEW
  SOFTWARE_VERSION_ID                     = “V1R2B”

OBJECT                                    = SOFTWARE_INFORMATION
  SOFTWARE_NAME                           = "NASAVIEW - PDS DATA PRODUCT ACCESS TOOL
                                                V1.2B"
  DATA_FORMAT                             = PDS
  SOFTWARE_LICENSE_TYPE                   = PUBLIC_DOMAIN
  TECHNICAL_SUPPORT_TYPE                  = FULL
  REQUIRED_STORAGE_BYTES                  = “1.8MB”
  PDS_USER_ID                             = SHUGHES
  NODE_ID                                 = CN
  SOFTWARE_DESC                           = “

  Software Overview
  =============
    NasaView Version 1.2b is a PDS Image display program developed for the following platforms:
        (a) PC / Win32
        (b) Unix / Sun OS

    NasaView is capable of accessing and displaying all images, tables, cubes, and histograms in the PDS archive.
    This release has been tested using Galileo, Magellan, Viking, MDIM, Voyager, IHW LSPN, and Clementine
    uncompressed images.

    NasaView is planned as a PDS data product object display utility that will run on SUN, MAC, and PC platforms
    in a GUI environment.

    This application was built using the Label Library Light (L3), Object Access Library (OAL), and the XVT
    Development Solution for C package. Label Library Light parses PDS ODL labels and creates an in-memory
    representation of the label information. The Object Access Library uses the parse-tree and accesses the actual
    PDS object. The XVT Development Solution supplies the cross platform GUI and an Object-oriented
    environment. XVT allows the definition of visual objects such as Windows and Menus and associates events
    and code with them.

  Available Support Material
  ==================
      BINARIES


   Programming Language
   ================
      SUN_C


   Platforms Supported
   ==============
      PC / Microsoft Win95, Win98, NT4.0


   Support Software Required / Used
   ========================
      X_WINDOWS

END_OBJECT                          = SOFTWARE_INFORMATION

OBJECT                              = SOFTWARE_ONLINE
  ON_LINE_IDENTIFICATION            = “http://pds.jpl.nasa.gov/license.html”
  ON_LINE_NAME                      = “NASAVIEW REVISION 2 BETA”
  NODE_ID                           = CN
  PROTOCOL_TYPE                     = URL
  PLATFORM                          = PC/WIN32
END_OBJECT                          = SOFTWARE_ONLINE

OBJECT                              = SOFTWARE_PURPOSE
  SOFTWARE_PURPOSE                  = DISPLAY
END_OBJECT                          = SOFTWARE_PURPOSE

END_OBJECT                          = SOFTWARE
END
```

B.11 TARGET

The TARGET catalog object forms part of a standard set for the submission of a target to the PDS. The TARGET object contains the following sub-objects: TARGET_INFORMATION and TARGET_REFERENCE_INFORMATION

(1) The TARGET INFORMATION catalog object provides target physical and dynamic parameters.

(2) The TARGET REFERENCE INFORMATION catalog object is completed for each reference associated with the target. If there is more than one reference, this object is repeated. A separate REFERENCE template is completed to provide the proper citation for each reference.

Example

```
/* Template: Target Template                    Rev: 1995-01-01                  */

/* Note:  The following template is used for the                                 */
/*        submission of a target to the PDS                                      */

OBJECT                                          = TARGET
TARGET_NAME                                     = JUPITER

OBJECT                                          = TARGET_INFORMATION
TARGET_TYPE                                     = PLANET
PRIMARY_BODY_NAME                               = SUN
ORBIT_DIRECTION                                 = PROGRADE
ROTATION_DIRECTION                              = PROGRADE
TARGET_DESC                                     = “

  A_AXIS_RADIUS : 71492.000000
  B_AXIS_RADIUS : 71492.000000
  BOND_ALBEDO  : UNK
  C_AXIS_RADIUS : 66854.000000
  FLATTENING : 0.006500
  MAGNETIC_MOMENT : 155000000000000000000.000000
  MASS : 1898799999999999953652202602496.000000
  MASS_DENSITY : 1.330000
  MINIMUM_SURFACE_TEMPERATURE : UNK
  MAXIMUM_SURFACE_TEMPERATURE : UNK
  MEAN_SURFACE_TEMPERATURE : UNK
  EQUATORIAL_RADIUS : 71492.000000
  MEAN_RADIUS : 69911.000000
  SURFACE_GRAVITY : 25.900000
  REVOLUTION_PERIOD : 4333.000000
  POLE_RIGHT_ASCENSION : 268.000000
  POLE_DECLINATION : 64.500000
  SIDEREAL_ROTATION_PERIOD : 0.410000
  MEAN_SOLAR_DAY : 0.410000
  OBLIQUITY : 3.100000
  ORBITAL_ECCENTRICITY : 0.048000
  ORBITAL_INCLINATION : 1.300000
  ORBITAL_SEMIMAJOR_AXIS : 778376719.000000
  ASCENDING_NODE_LONGITUDE : 100.500000
```

```
    PERIAPSIS_ARGUMENT_ANGLE : 275.200000”

END_OBJECT                                    = TARGET_INFORMATION

OBJECT                                        = TARGET_REFERENCE_INFORMATION
REFERENCE_KEY_ID                              = “XYZ95”
END_OBJECT                                    = TARGET_REFERENCE_INFORMATION

END_OBJECT                                    = TARGET
END
```

APPENDIX C

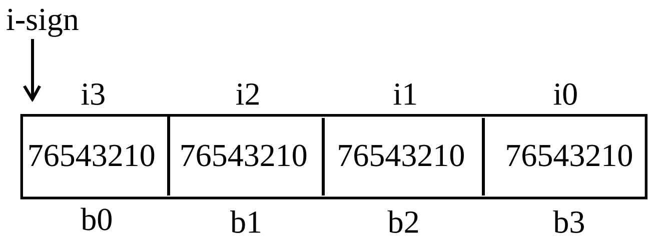
Internal Representation of Data Types

This appendix contains the detailed internal representations of the PDS standard data types listed in Table 3.2 of the Data Type Definitions chapter of this document.

C.1 MSB_INTEGER

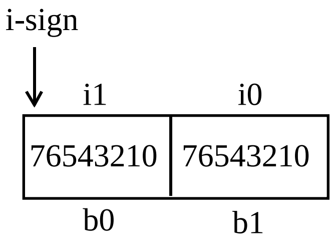
Aliases: INTEGER, MAC_INTEGER, SUN_INTEGER

MSB 4-byte integers:



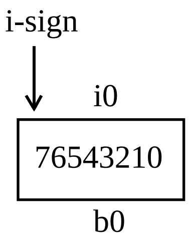
* Bit 7 in i3 is used for the sign bit.

MSB 2-byte integers:



* Bit 7 in i1 is used for the sign bit.

MSB 1-byte integers:



* Bit 7 is used for the sign bit.

Where:

b0 - b3 = Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, and b3).

i-sign = integer sign bit

i0 - i3 = Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left, (e.g., lowest value =bit 0, highest value = bit 7) in the following way:

4-bytes:

In i0, bits 0-7 represent 2**0 through 2**7
In i1, bits 0-7 represent 2**8 through 2**15
In i2, bits 0-7 represent 2**16 through 2**23
In i3, bits 0-6 represent 2**24 through 2**30

2-bytes:

In i0, bits 0-7 represent 2**0 through 2**7
In i1, bits 0-6 represent 2**8 through 2**14

1-byte:

In i0, bits 0-6 represent 2**0 through 2**6

All negative signed values are assumed to be twos-compliment.

C.2 MSB_UNSIGNED_INTEGER

Aliases:MAC_UNSIGNED_INTEGER, SUN_UNSIGNED_INTEGER, UNSIGNED_INTEGER

MSB 4 byte unsigned integers:

i3	i2	i1	i0
76543210	76543210	76543210	76543210
b0	b1	b2	b3

MSB 2-byte unsigned integers:

i1	i0
76543210	76543210
b0	b1

MSB 1-byte unsigned integers:

i0
76543210
b0

Where:

b0 - b3 = Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, and b3).

i0 - i3 = Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left, (e.g., lowest value =bit 0, highest value = bit 7) in the following way:

4-bytes:
In i0, bits 0-7 represent 2**0 through 2**7
In i1, bits 0-7 represent 2**8 through 2**15
In i2, bits 0-7 represent 2**16 through 2**23
In i3, bits 0-7 represent 2**24 through 2**31

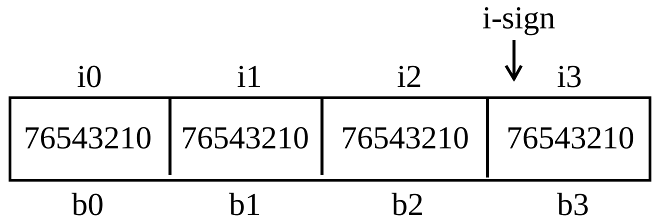
2-bytes:
In i0, bits 0-7 represent 2**0 through 2**7
In i1, bits 0-7 represent 2**8 through 2**15

1-byte:
In i0, bits 0-7 represent 2**0 through 2**7

C.3 LSB_INTEGER

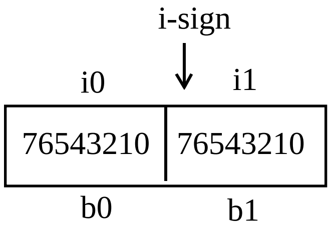
Aliases: PC_INTEGER, VAX_INTEGER

LSB 4-byte integers:



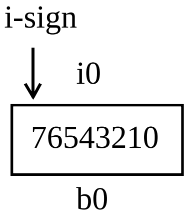
* Bit 7 in i3 is used for the sign bit.

LSB 2-byte integers:



* Bit 7 in i1 is used for the sign bit.

LSB 1-byte integers:



* Bit 7 in i1 is used for the sign bit.

Where:

b0 - b3 = Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, and b3).

i-sign = integer sign bit

i0 - i3 = Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left, (e.g., lowest value =

bit 0, highest value = bit 7) in the following way:

4-bytes:
In i0, bits 0-7 represent 2**0 through 2**7
In i1, bits 0-7 represent 2**8 through 2**15
In i2, bits 0-7 represent 2**16 through 2**23
In i3, bits 0-6 represent 2**24 through 2**30

2-bytes:
In i0, bits 0-7 represent 2**0 through 2**7
In i1, bits 0-6 represent 2**8 through 2**14

1-byte:
In i0, bits 0-6 represent 2**0 through 2**6

All negative signed values are assumed to be twos-compliment.

C.4 LSB_UNSIGNED_INTEGER

Aliases: PC_UNSIGNED_INTEGER, VAX_UNSIGNED_INTEGER

LSB 4-byte unsigned integers:

i0	i1	i2	i3
76543210	76543210	76543210	76543210
b0	b1	b2	b3

LSB 2-byte unsigned integers:

i0	i1
76543210	76543210
b0	b1

LSB 1-byte unsigned integers:

i0
76543210
b0

Where:

b0 - b3 = Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, and b3).

i0 - 13 =Arrangement of bytes in the integer, from lowest order to highest order. The bits within each byte are interpreted from right to left, (e.g., lowest value =bit 0, highest value = bit 7) in the following way:

4-bytes:

In i0, bits 0-7 represent 2**0 through 2**7
In i1, bits 0-7 represent 2**8 through 2**15
In i2, bits 0-7 represent 2**16 through 2**23
In i3, bits 0-7 represent 2**24 through 2**31

2-bytes:
In i0, bits 0-7 represent 2**0 through 2**7
In i1, bits 0-7 represent 2**8 through 2**15

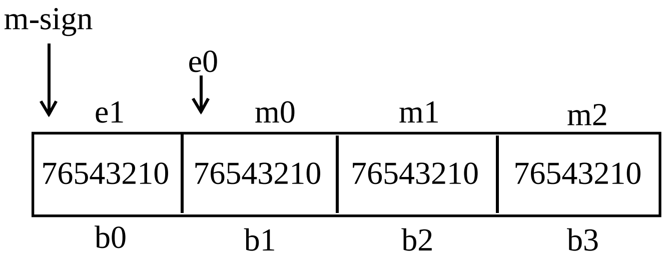
1-byte:

In i0, bits 0-7 represent 2**0 through 2**7

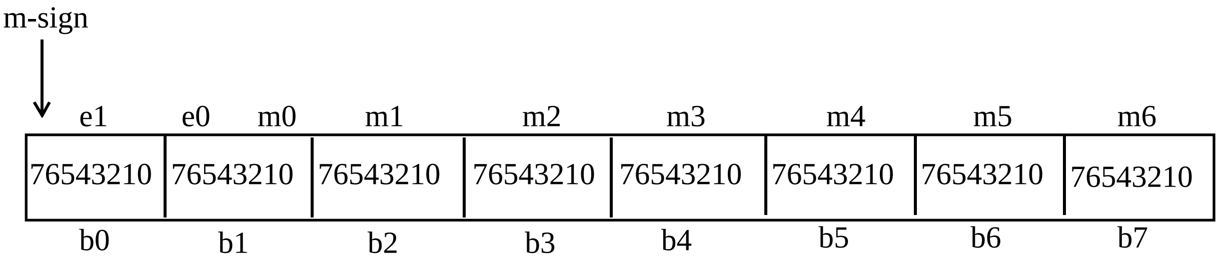
C.5 IEEE_REAL

Aliases: FLOAT, MAC_REAL, REAL, SUN_REAL

IEEE 4-byte real numbers:

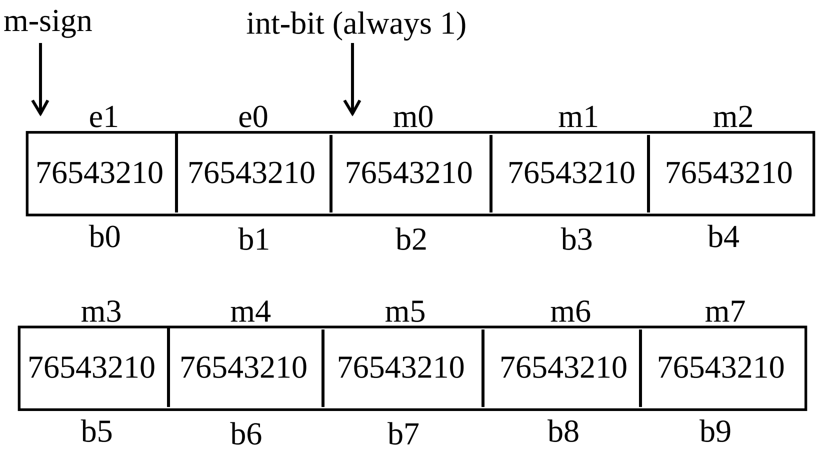


IEEE 8-byte (double precision) real numbers:



* Bit 7 in e1 is used for the mantissa sign bit.

IEEE 10-byte (temporary) real numbers:



* Bit 7 in e1 is used for the mantissa sign bit.

Where:

b0 - b9 = Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, b3, etc.).

m-sign = Mantissa sign bit

int-bit = In l0 byte reals only, the implicit "1" is actually specified by this bit.

e0 - e1 = Arrangement of the portions of the bytes that make up the exponent, from lowest order to highest order. The bits within each byte are interpreted from right to left, (e.g.,lowest value = rightmost bit in the exponent part of the byte, highest value = leftmost bit in the exponent part of the byte) in the following way:

4-bytes (single precision):

In e0, bit 7 represents 2**0
In e1, bits 0-6 represent 2**1 through 2**7

Exponent bias = 127

8-bytes (double precision):

In e0, bits 4-7 represent 2**0 through 2**3
In e1, bits 0-6 represent 2**4 through 2**10

Exponent bias = 1023

10-bytes (temporary):
In e0, bits 0-7 represent 2**0 through 2**7
In e1, bits 0-6 represent 2**8 through 2**14

Exponent bias = 16383

m0 - m7 = Arrangement of the portions of the bytes that make up the mantissa, from highest order fractions to the lowest order fractions. The order of the bits within each byte progresses from left to right, with each bit representing a fractional power of two, in the following way:

4 -bytes (single precision):
In m0, bits 6-0 represent 1/2**1 through 1/2**7
In m1, bits 7-0 represent 1/2**8 through 1/2**15
In m2, bits 7-0 represent 1/2**16 through 1/2**23

8-bytes (double precision):
In m0, bits 3-0 represent 1/2**1 through 1/2**4
In m1, bits 7-0 represent 1/2**5 through 1/2**12
In m2, bits 7-0 represent 1/2**13 through 1/2**20
In m3, bits 7-0 represent 1/2**21 through 1/2**28
In m4, bits 7-0 represent 1/2**29 through 1/2**36
In m5, bits 7-0 represent 1/2**37 through 1/2**44
In m6, bits 7-0 represent 1/2**45 through 1/2**52

10-bytes (temporary):
In m0, bits 6-0 represent 1/2**1 through 1/2**7
In m1, bits 7-0 represent 1/2**8 through 1/2**15
In m2, bits 7-0 represent 1/2**16 through 1/2**23
In m3, bits 7-0 represent 1/2**24 through 1/2**31
In m4, bits 7-0 represent 1/2**32 through 1/2**39
In m5, bits 7-0 represent 1/2**40 through 1/2**47
In m6, bits 7-0 represent 1/2**48 through 1/2**55
In m7, bits 7-0 represent 1/2**56 through 1/2**63

These representations all follow the format:

1. (mantissa) x 2** (exponent - bias)

with the "1." part implicit (except for the 10-byte temp real, in which the "1." part is actually stored in the third byte (b2)), In all cases, the exponent is stored as an unsigned, biased integer (e.g., exponent-as-stored - bias = true exponent value).

C.6 IEEE_COMPLEX

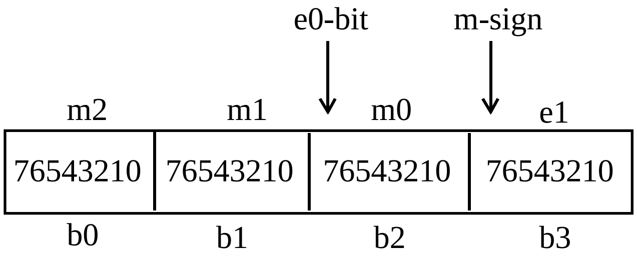
Aliases: COMPLEX, MAC_COMPLEX, SUN_COMPLEX

Two contiguous IEEE_REALs in memory, representing the real and imaginary parts.

C.7 PC_REAL

Aliases: None

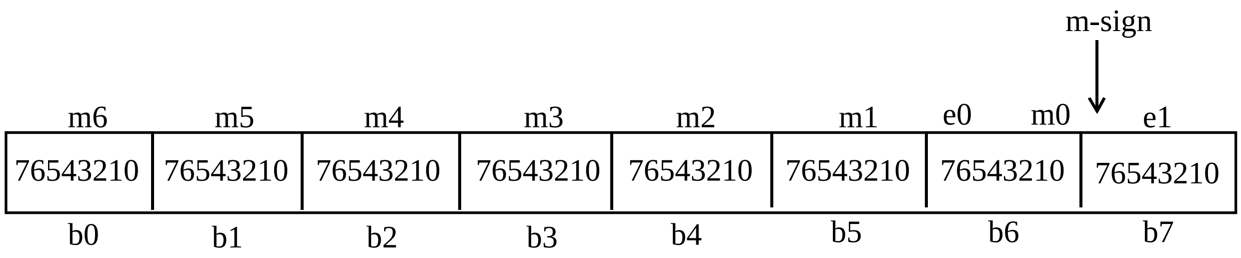
PC 4-byte real numbers:



m2	m1	m0	e1
76543210	76543210	76543210	76543210
b0	b1	b2	b3

* Bit 7 in e1 is used for the mantissa sign bit.

PC 8-byte (double precision) real numbers:



m6	m5	m4	m3	m2	m1	e0 m0	e1
76543210	76543210	76543210	76543210	76543210	76543210	76543210	76543210
b0	b1	b2	b3	b4	b5	b6	b7

* Bit 7 in e1 is used for the mantissa sign bit.

PC 10-byte (temporary) real numbers:

m7	m6	m5	m4	m3
76543210	76543210	76543210	76543210	76543210
b0	b1	b2	b3	b4

		int-bit ↓		m-sign ↓
m2	m1	m0	e0	e1
76543210	76543210	76543210	76543210	76543210
b5	b6	b7	b8	b9

Where:

b0 - b9 = Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, b3, etc.).

m-sign = Mantissa sign bit

int-bit = In 10 byte reals only, the implicit "1" is actually specified by this bit.

e0 - e1 = Arrangement of the portions of the bytes that make up the exponent, from lowest order to highest order. The bits within each byte are interpreted from right to left, (e.g., lowest value = rightmost bit in the exponent part of the byte, highest value = leftmost bit in the exponent part of the byte) in the following way:

4-bytes (single precision) :
In e0, bit 7 represents 2**0
In e1, bits 0-6 represent 2**1 through 2**7

Exponent bias = 127

8-bytes (double precision) :
In e0, bits 4-7 represent 2**0 through 2**3
In e1, bits 0-6 represent 2**4 through 2**10

Exponent bias = 1023

10-bytes (temporary):
In e0, bits 0-7 represent 2**0 through 2**7
In e1, bits 0-6 represent 2**4 through 2**10

Exponent bias = 16383

m0 - m7 = Arrangement of the portions of the bytes that make up the mantissa, from highest order fractions to lowest order fractions. The order of the bits within each byte progresses from left to right, with each bit representing a fractional power of two, in the following way:

4-bytes (single precision) :
In m0, bits 6-0 represent 1/2**1 through 1/2**7
In m1, bits 7-0 represent 1/2**8 through 1/2**15
In m2, bits 7-0 represent 1/2**16 through 1/2**23

8-bytes (double precision) :
In m0, bits 3-0 represent 1/2**1 through 1/2**4
In m1, bits 7-0 represent 1/2**5 through 1/2**12
In m2, bits 7-0 represent 1/2**13 through 1/2**20
In m3, bits 7-0 represent 1/2**21 through 1/2**28
In m4, bits 7-0 represent 1/2**29 through 1/2**36
In m5, bits 7-0 represent 1/2**37 through 1/2**44
In m6, bits 7-0 represent 1/2**45 through 1/2**52

10-bytes (temporary) :
In m0, bits 6-0 represent 1/2**1 through 1/2**7
In m1, bits 7-0 represent 1/2**8 through 1/2**15
In m2, bits 7-0 represent 1/2**16 through 1/2**23
In m3, bits 7-0 represent 1/2**24 through 1/2**31
In m4, bits 7-0 represent 1/2**32 through 1/2**39
In m5, bits 7-0 represent 1/2**40 through 1/2**47
In m6, bits 7-0 represent 1/2**48 through 1/2**55
In m7, bits 7-0 represent 1/2**56 through 1/2**63

These representations all follow the format:

1. (mantissa) x 2**(exponent - bias)

with the "1." part implicit (except for the 10-byte temp real, in which the "1." part is actually stored in the third byte (b2)),

In all cases, the exponent is stored as an unsigned, biased integer (e.g., exponent-as-stored - bias=true exponent value).

C.8 PC_COMPLEX

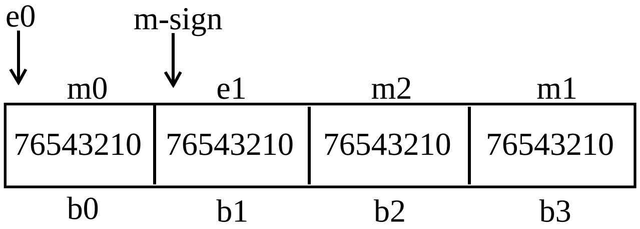
Aliases: None

Two contiguous PC-REALs in memory, representing the real and imaginary parts.

C.9 VAX_REAL, VAXG_REAL

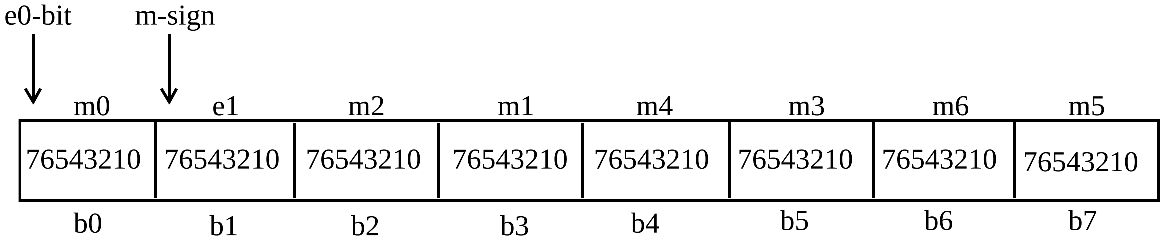
Aliases: VAX_DOUBLE (for VAX_REAL only, none for VAXG_REAL)

VAX F-type 4-byte real numbers:



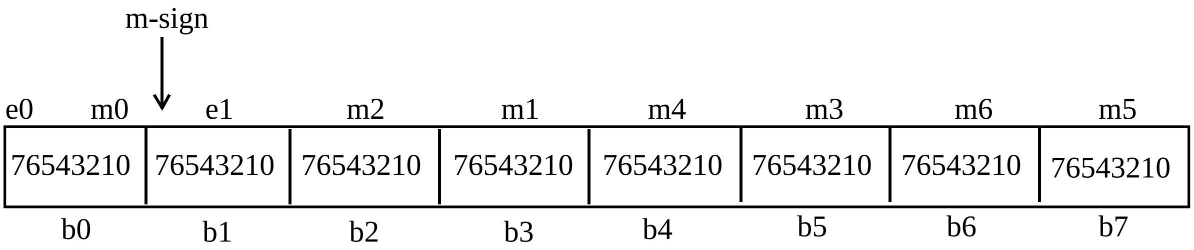
* Bit 7 in e1 is used for the mantissa sign bit.

VAX D-type 8-byte real numbers:

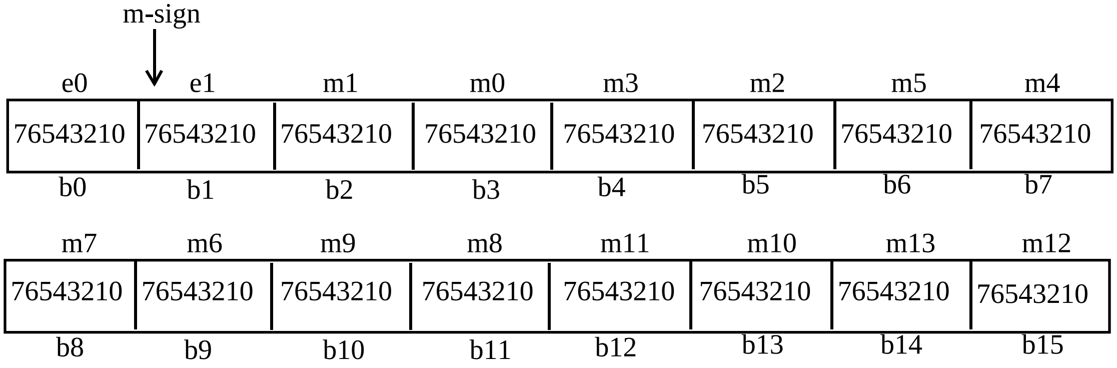


* Bit 7 in e1 is used for the mantissa sign bit.

VAX G-type 8-byte real numbers:



VAX H-type 16-byte real numbers:



Where:

b0 - b15 = Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, b3, etc.).

m-sign = Mantissa sign bit

e0 - e1 = Arrangement of the portions of the bytes that make up the exponent, from lowest order to highest order. The bits within each byte are interpreted from right to left, (e.g., lowest value= rightmost bit in the exponent part of the byte, highest value = leftmost bit in the exponent part of the byte) in the following way:

4-bytes (F-type, single precision) :
In e0, bit 7 represents 2**0
In e1, bits 0-6 represent 2**1 through 2**7

Exponent bias = 129

8-bytes (D-type, double precision) :
In e0, bit 7 represents 2**0
In e1, bits 0-6 represent 2**1 through 2**7

Exponent bias = 129

8-bytes (G-type, double precision) :
In e0, bits 4-7 represent 2**0 through 2**3
In e1, bits 0-6 represent 2**4 through 2**10

Exponent bias = 1025

16-bytes (H-type) :

In e0, bits 0-7 represent 2**0 through 2**7
In e1, bits 0-6 represent 2**8 through 2**14

Exponent bias = 16385

m0 -m13 = Arrangement of the portions of the bytes that make up the mantissa, from highest order fractions to lowest order fractions. The order of the bits within each byte progresses from left to right, with each bit representing a fractional power of two, in the following way:

4-bytes (F-type, single precision) :
In m0, bits 6-0 represent 1/2**1 through 1/2**7
In m1, bits 7-0 represent 1/2**8 through 1/2**15
In m2, bits 7-0 represent 1/2**16 through 1/2**23

8-bytes (D-type, double precision) :
In m0, bits 6-0 represent 1/2**1 through 1/2**7
In m1, bits 7-0 represent 1/2**8 through 1/2**15
In m2, bits 7-0 represent 1/2**16 through 1/2**23
In m3, bits 7-0 represent 1/2**24 through 1/2**31
In m4, bits 7-0 represent 1/2**32 through 1/2**39
In m5, bits 7-0 represent 1/2**40 through 1/2**47
In m6, bits 7-0 represent 1/2**48 through 1/2**55

8-bytes (G-type, double precision) :
In m0, bits 3-0 represent 1/2**1 through 1/2**4
In m1, bits 7-0 represent 1/2**5 through 1/2**12
In m2, bits 7-0 represent 1/2**13 through 1/2**20
In m3, bits 7-0 represent 1/2**21 through 1/2**28
In m4, bits 7-0 represent 1/2**29 through 1/2**36
‘ In m5, bits 7-0 represent 1/2**37 through 1/2**44
In m6, bits 7-0 represent 1/2**45 through 1/2**52

16-bytes (H-type) :
In m0, bits 7-0 represent 1/2**1 through 1/2**8
In m1, bits 7-0 represent 1/2**9 through 1/2**16
In m2, bits 7-0 represent 1/2**17 through 1/2**24
In m3, bits 7-0 represent 1/2**25 through 1/2**32
In m4, bits 7-0 represent 1/2**33 through 1/2**40
In m5, bits 7-0 represent 1/2**41 through 1/2**48
In m6, bits 7-0 represent 1/2**49 through 1/2**56
In m7, bits 7-0 represent 1/2**57 through 1/2**64
In m8, bits 7-0 represent 1/2**65 through 1/2**72
In m9, bits 7-0 represent 1/2**73 through 1/2**80

In m10, bits 7-0 represent 1/2**81 through 1/2**88
In m11, bits 7-0 represent 1/2**89 through 1/2**96
In m12, bits 7-0 represent 1/2**97 through 1/2**104
In m13, bits 7-0 represent 1/2**105 through 1/2**112

These representations all follow the format:

1. (mantissa) x 2**(exponent - bias)

with the "1." part implicit

In all cases, the exponent is stored as an unsigned, biased integer (e.g., exponent-as-stored - bias = true exponent value).

C.10 VAX_COMPLEX, VAXG_COMPLEX

Aliases: None

Two contiguous VAX_REALs or VAXG_REALs in memory, representing the real and imaginary parts.

C.11 MSB_BIT_STRING

Aliases: BIT_STRING

MSB n-byte bit strings:

As read from a file:

bits 1-8	bits 9-16	bits 17-24	bits 25-32		bits ((nx8)-7) - (nx8)
76543210	76543210	76543210	76543210	. . .	
b0	b1	b2	b3		b x (n-1)

No byte swapping is needed.
Note: for n-byte bitstrings, continue pattern above.

MSB 2-byte bit strings:

As read from file:

```
 bits       bits
 1-8        9-16
+----------+----------+
|76543210  |76543210  |
+----------+----------+
    b0         b1
```

No byte swapping is needed.

MSB 1-byte bit strings:

As read from file:

```
 bits
 1-8
+----------+
|76543210  |
+----------+
    b0
```

No byte swapping is needed.

Where:

b0 - b3 =Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, and b3).

The bits within a byte are numbered from left to right:

```
76543210
↑       ↑
bit 1   bit 8
```

C.12 LSB_BIT_STRING

Aliases: VAX_BIT_STRING

LSB 4-byte bit strings:

As read from a file:

bits 25-32	bits 17-24	bits 9-16	bits 1-8
76543210	76543210	76543210	76543210
b0	b1	b2	b3

After bytes are swapped:

bits 1-8	bits 9-16	bits 17-24	bits 25-32
76543210	76543210	76543210	76543210
b3	b2	b1	b0

LSB 2-byte bit strings:

As read from a file:

bits 9-16	bits 1-8
76543210	76543210
b0	b1

After bytes are swapped:

bits 1-8	bits 9-16
76543210	76543210
b1	b0

LSB 1-byte bit strings:

As read from file:

bits 1-8
76543210
b0

No byte swapping is needed.

Where:

b0 - b3 =Arrangement of bytes as they appear when read from a file (e.g., read b0 first, then b1, b2, and b3).

The bits within a byte are numbered from left to right:

```
  76543210
  ↑      ↑
bit 1   bit 8
```

APPENDIX D

Examples of Required Files

The examples in this Appendix are based on both existing or planned PDS archive volumes, but have been modified to reflect the most recent version of the PDS standards.

D.1 AAREADME.TXT

Each PDS archive volume shall include an "AAREADME.TXT" file that contains an overview of the contents and structure of the volume. An annotated outline is provided here as guidance for compiling this file.

Annotated Outline

I. PDS TEXT Object (must appear in an attached or detached label)

II. Volume Title

III. Contents

1. Introduction
    - a. Science data content
    - b. Conformance to PDS standards
    - c. Document or institutional references for additional science information

2. Volume format
    - a. Computer systems that can access the volume
    - b. International standards to which the volume conforms

3. File formats
    - a. Data record formats
    - b. Specifications for specialized files (e.g., Postscript)
    - c. Description of PDS objects, pointers, etc.

4. Volume contents
    - a. Directory structure of the volume

5. Recommended CD-ROM drives (if applicable)
    - a. Driver descriptions and notes for all appropriate computer platforms

6. Errata (if applicable)
    a. Known errors, cautionary notes, disclaimers, etc.
    b. Reference to the ERRATA.TXT file on the volume or online

7. Contacts
    a. Names and addresses of people or organizations to contact for questions concerning science data, technical support, data product generation and labelling, etc.

Example:

The following is an example of an AAREADME.TXT file used on a PDS archive volume that does not use the logical volume construct. Note that section 3 in the example would need to be updated if logical volumes were present.

```
PDS_VERSION_ID                    = PDS3

RECORD_TYPE                       = FIXED_LENGTH
RECORD_BYTES                      = 80
SPACECRAFT_NAME                   = MAGELLAN
TARGET_NAME                       = VENUS
OBJECT                            = TEXT
PUBLICATION_DATE                  = 1994-06-01
NOTE                              = "MAGELLAN LOSAPDR ARCHIVE CD-WO"
END_OBJECT                        = TEXT
END
```

MAGELLAN LOSAPDR ARCHIVE CD-WO

1. Introduction

This CD-WO contains Magellan Cycle 4 LOSAPDR (Line of Sight Acceleration Profile Data Record) products. It also contains documentation which describe the LOSAPDRs. Each LOSAPDR product contains the results from processing of radio tracking data of the Magellan spacecraft. There are 866 LOSAPDRs on this volume.

The LOSAPDR products archived on this volume are the exact products released by the Magellan Project. Supporting documentation and label files conform to the Planetary Data System (PDS) Standards, Version 3.0, Jet Propulsion Laboratory (JPL) document JPL D-7669.

Additional information about the Magellan gravity experiment, including the acquisition, processing, and quality of the LOSAPDR data, can be found in JPL documents that are available from the PDS Geosciences Node, Washington University, St. Louis, MO.

2. Disk Format

The disk has been formatted so that a variety of computer systems (e.g. IBM PC, Macintosh, Sun) may access the data. Specifically, it is formatted according to the ISO 9660 level 1 Interchange Standard. For further information, refer to the ISO 9660 Standard Document: RF# ISO 9660-1988, 15 April 1988.

3. File Formats

Each orbit for which gravity data exists is represented by one LOSAPDR data file. The LOSAPDR is an ASCII file. The data file contains 3 tables: 1) HEADER_TABLE; 2) TIMES_TABLE; and 3) RESULTS_TABLE. The HEADER_TABLE is a single-row multi-column table containing information on initial values, control parameters, and simple calculations required by the program that generates the data files. The TIMES_TABLE is a single column containing exact times bounding spline intervals to the Doppler residuals. The number of rows is variable. The RESULTS_TABLE contains the results from spline fits to Doppler residuals. Each row in the table contains times, Doppler residuals, spacecraft position and velocity information, and inferred spacecraft acceleration. The data files are described by PDS labels embedded at the beginning of the file. Further information on LOSAPDR file formats and contents can also be obtained from the Magellan Software Interface Specification (SIS) document NAV-138. A copy of the document is stored on this disk as file LOSAPDR.TXT in the DOCUMENT directory.

All document files and detached label files contain 80-byte fixed-length records, with a carriage return character (ASCII 13) in the 79th byte and a line feed character (ASCII 10) in the 80th byte. This allows the files to be read by the MacOS, DOS, Unix, and VMS operating systems. All tabular files are also described by PDS labels, either embedded at the beginning of the file or detached. If detached, the PDS label file has the same name as the data file it describes, with the extension .LBL; for example, the file INDEX.TAB is accompanied by the detached label file INDEX.LBL in the same directory.

Tabular files are formatted so that they may be read directly into many database management systems on various computers. All fields are separated by commas, and character fields are enclosed in double quotation marks ("). Character fields are left justified, and numeric fields are right justified. The "start byte" and "bytes" values listed in the labels do not include the commas between fields or the quotation marks surrounding character fields. The records are of fixed length, and the last two bytes of each record contain the ASCII carriage return and line feed characters. This allows a table to be treated as a fixed length record file on computers that support this file type and as a normal text file on other computers.

A PostScript file, REPORT.PS, is included on this volume. This PostScript document is a validation report that lists all LOSAPDRs, and gives specific information, comments, and the status of each data file after a quality check and validation at the PDS Geophysics Subnode. The document is described by the detached label file, REPORT.LBL. The document can also be viewed by a Display PostScript program and can be printed out from a PostScript printer. The ASCII text version of the PostScript file is REPORT.ASC.

PDS labels are object-oriented. The object to which the label refers (e.g., IMAGE, TABLE, etc.) is denoted by a statement of the form:

```
^object = location
```

in which the carat character (^, also called a pointer in this context) indicates that the object starts at the given location. In an attached label, the location is an integer representing the starting record number of the object (the first record in the file is record 1). In a detached label, the location denotes the name of the file containing the object, along with the starting record or byte number. For example:

```
^TABLE = "INDEX.TAB"
```

indicates that the TABLE object points to the file INDEX.TAB .

Pointers to data objects are always required to be located in the same directory as the label file, so the file INDEX.TAB in this example is located in the same directory as the detached label file.

Other types of pointer statements can also be found on this volume. To resolve the pointer statement, first look in the same directory as the file containing the pointer statement. If the pointer is still unresolved, look in the following top level directory:
^ STRUCTURE - LABEL directory
^ CATALOG - CATALOG directory
^DATA_SET_MAP_PROJECTION - CATALOG directory
^DESCRIPTION - DOCUMENT directory.

Below is a list of the possible formats for the ^object keyword.

```
^object = n
^object = n<BYTES>
^object = "filename.ext"
^object = ("filename.ext",n)
^object = ("filename.ext",n<BYTES>)
```

where

n is the starting record or byte number of the object, counting from the beginning of the file (record 1, byte 1)

<BYTES> indicates that the number given is in units of bytes

filename is the upper-case file name

ext is the upper-case file extension

4. CD-ROM Contents

The files on this CD-ROM are organized in one top-level directory with several subdirectories. The following table shows the structure and content of these directories. In the table, directory names are enclosed in square brackets ([]), upper-case letters indicate an actual directory or file name, and lower-case letters indicate the general form of a set of directory or file names.

```
FILE                                    CONTENTS

Top-level directory
|
|- AAREADME.TXT                         The file you are reading.
|
|- ERRATA.TXT                           Description of known anomalies and errors
|                                       present on this volume.
|
|- VOLDESC.CAT                          A description of the contents of this CD-
|                                       ROM volume in a format readable
|                                       by both humans and computers.
|
|- [CATALOG]                            A directory containing information about the
|       |                               LOSAPDR dataset.
|       |
|       |- CATALOG.CAT                  PDS catalog objects. Mission, spacecraft
|       |                               and instrument descriptions.
|       |
|       |- CATINFO.TXT                  Description of files in the CATALOG
|       |                               directory.
|       |
|       |- DATASET.CAT                  PDS dataset catalog object. A description
|                                       of the dataset, parameters, processing, data
|                                       coverage and quality.
|
|- [DATA]                               A directory containing LOSAPDR data files.
|       |- [mmmmnnnn]                   Directories containing LOSAPDR data files
|       |                               for orbits between 'mmmm' and 'nnnn'.
|       |
|       |       |- L0mmmm.001           LOSAPDR file for orbit number 'mmmm'.
|
|- [DOCUMENT]                           A directory containing document files
|                                        relating to this disk.
|       |
|       |- DOCINFO.TXT                  Description of files in the DOCUMENT
|       |                               directory.
|       |
|       |- LOSAPDR.TXT                  A machine readable version of the LOSAPDR
|       |                               SIS document describing the format and
|       |                               content of the data files.
|       |
|       |- REPORT.ASC                   ASCII text version of REPORT.PS.
|       |
|          |- REPORT.LBL                    A PDS detached label describing REPORT.ASC & REPORT.PS.
|          |
|          |- REPORT.PS                     A PostScript document that gives specific
|                                           information about each LOSAPDR after a
|                                           quality check and validation.
|
|- [INDEX]                                  A directory containing index files relating
|          |                                to this disk.
|          |
|          |- INDEX.LBL                     A PDS detached label describing INDEX.TAB.
|          |
|          |- INDEX.TAB                     Tabular summary of data files.
|          |
|          |- INDXINFO.TXT                  Description of files in the INDEX directory.
```

5. Recommended CD-ROM Drives and Driver Software

VAX/VMS
Drive: Digital Equipment Corporation (DEC) RRD40 or RRD50. Driver: DEC VFS CD-ROM driver V4.7 or V5.2 and up.

Note: The driver software may be obtained from Jason Hyon at JPL. It is necessary to use this driver to access Extended Attribute Records (XARs) on a CD-ROM.

VAX/Ultrix
Drive: DEC RRD40 or RRD50. Driver: Supplied with Ultrix 3.1.

Note: Internet users can obtain a copy of the "cdio" software package via anonymous ftp from the "space.mit.edu" server in the file named "src/cdio.shar". Contact Dr. Peter Ford at Massachusetts Institute of Technology for details (617-253-6485 or pgf@space.mit.edu).

IBM PC
Drive: Toshiba, Hitachi, Sony, or compatible. Driver: Microsoft MSCDEX version 2.2.

Note: The latest version of MSCDEX (released in February 1990) is generally available. Contact Jason Hyon for assistance in locating a copy.

Apple Macintosh
Drive: Apple CD SC (Sony) or Toshiba. Driver: Apple CD-ROM driver.

Note: The Toshiba drive requires a separate driver, which may be obtained from Toshiba.

Sun Micro (SunOS 4.0.x and earlier)
Drive: Delta Microsystems SS-660 (Sony). Driver: Delta Microsystems driver or SUN sr.o Driver.

Note: For questions concerning this driver, contact Denis Down at Delta Microsystems, 415-449-6881.

Sun Micro (SunOS 4.0.x and later)
Drive: Sun Microsystems. Driver: SunOS sr.o driver.

Note: A patch must be made to SunOS before the Sun driver can access any CD-ROM files containing Extended Attribute Records. A copy of this patch is available to Internet

users via anonymous ftp from the "space.mit.edu" server
in the file named "src/SunOS.4.x.CD-ROM.patch".

6. Errata and Disclaimer

A cumulative list of anomalies and errors is maintained in the file ERRATA.TXT at the root directory of this volume.

Although considerable care has gone into making this volume, errors are both possible and likely. Users of the data are advised to exercise the same caution as they would when dealing with any other unknown data set.

Reports of errors or difficulties would be appreciated. Please contact one of the persons listed herein.

7. Whom to Contact for Information

For questions concerning this volume set, data products and documentation:

Jim Alexopoulos Washington University Dept. of Earth and Planetary Sciences 1 Brookings Drive Campus Box 1169
St. Louis, MO 63130 314-935-5365

Electronic mail address: Internet: jim@wuzzy.wustl.edu

For questions about how to read the CD-ROM:

Jason J. Hyon Jet Propulsion Laboratory California Institute of Technology 4800 Oak Grove Drive MS 525-3610
Pasadena, CA 91109 818-306-6054

Electronic mail addresses: Internet: jhyon@jplpds.jpl.nasa.gov NASAmail: JHYON NSI: JPLPDS::JHYON X.400: (ID:JHYON,PRMD:NASAMAIL,ADMD:TELEMAIL,C:USA)

For questions concerning the generation of LOSAPDR products:

William L. Sjogren Magellan Gravity Principal Investigator Jet Propulsion Laboratory California Institute of Technology
4800 Oak Grove Drive MS 301-150 Pasadena, CA 91109 818-354-4868

Electronic mail address: Internet: wls@nomad.jpl.nasa.gov

For questions concerning LOSAPDR data:

William L. Sjogren
Jet Propulsion Laboratory Pasadena, CA

Dr. Roger J. Phillips Washington University Dept. of Earth and Planetary Sciences 1 Brookings Dr. Campus Box 1169
St. Louis, MO 63130 314-935-6356

Electronic mail address: Internet: phillips@wustite.wustl.edu

For questions concerning LOSAPDR labels:

Dr. Richard Simpson Stanford University Durand Bldg. Room 232 Stanford, CA 94305-4055 415-723-3525

Electronic mail address: Internet: rsimpson@magellan.stanford.edu

This disk was produced by Jim Alexopoulos.

D.2 INDXINFO.TXT

Each PDS archive volume shall include an “INDXINFO.TXT” file in the INDEX subdirectory that contains an overview of the contents and structure of the index table or tables on the volume as well as usage notes. An example is provided here as guidance for compiling this file.

Example:

```
CCSD3ZF0000100000001NJPL3IF0PDSX00000001
PDS_VERSION_ID                          = PDS3

RECORD_TYPE                             = STREAM
OBJECT                                  = TEXT
NOTE                                    = "Notes on using the image index tables."
PUBLICATION_DATE                        = 1990-12-20
END_OBJECT                              = TEXT
END
```

NOTES ON USING THE IMAGE INDEX TABLES

These notes describe the contents and format of the two image index tables on this CD-ROM, INDEX.TAB and CUMINDEX.TAB.

The image index table (INDEX.TAB) contains one record for each image file on this Viking Orbiter CD-ROM. The cumulative image index table (CUMINDEX.TAB) contains one record for each image file on all the Viking Orbiter CD-ROMs published so far. The following description applies to both of these tables.

The image index tables are formatted so that they may be read directly into many database management systems on various computers.

All fields are separated by commas, and character fields are enclosed in double quotation marks ("). Each record contains 512 bytes of ASCII character data (1 character = 1 byte). Bytes 511 and 512 contain the ASCII carriage return and line feed characters. This allows the table to be treated as a fixed length record file on computers that support this file type and as a normal text file on other computers. The structure and content of the image index tables are described in the file VOLINFO.TXT located in the DOCUMENT directory. The files INDEX.LBL and CUMINDEX.LBL contain labels for INDEX.TAB and CUMINDEX.TAB coded in the Object Description Language (ODL), providing a formal description of the index table structure.

Users of most commercial database management systems should be able to use the list below to define the names and characteristics of each field and then to load the tables into their systems using a delimited ASCII text input format. If necessary the specific column start positions and lengths can be used to load the data.

For personal computer users, DBASE III DBF structures are also provided in the files INDEX.DBF and CUMINDEX.DBF. These files can be used to load the INDEX.TAB or CUMINDEX.TAB files into DBASE III or IV with the following commands:

```
USE INDEX
APPEND FROM INDEX.TAB DELIMITED

USE CUMINDEX
APPEND FROM CUMINDEX.TAB DELIMITED
```

Once the table is loaded into DBASE III, it can generally be automatically loaded into other data managers or spreadsheets that provide search and retrieval capabilities.

D.3 SOFTINFO.TXT

Each PDS archive volume that contains software (in the SOFTWARE subdirectory) shall include a “SOFTINFO.TXT” file. This file contains a description of the software and usage information. An outline and example are provided here as guidance for compiling this file.

Outline

I. PDS TEXT Object (must appear in an attached or detached label)

II. Contents

1. Introduction

2. Software Description
   A brief description of software included on the volume. This can be broken down into separate sections for each type of software. This should indicate where the software and its documentation reside in the software hierarchy, as well as describe any known limitations or problems.

3. Software Directory Structure (optional)

4. Software License Information and Disclaimers (if appropriate)

Example:

```
PDS_VERSION_ID                    = PDS3

RECORD_TYPE                       = FIXED_LENGTH
RECORD_BYTES                      = 80
OBJECT                            = TEXT
INTERCHANGE_FORMAT                = ASCII
PUBLICATION_DATE                  = 1994-10-01
NOTE                              = “Description of software provided with the Clementine CD-ROM
set”
END_OBJECT                        = TEXT
END
```

```
          Clementine  Software

1. Introduction

This directory contains software that provides display and processing
capabilities for the Clementine data archived on this CD-ROM set.
```

2. Software Description

2.1. Decompression Software

The PCDOS, MACSYS7 and SUNOS subdirectories all contain software which can be used to decompress the Clementine raw images. CLEMDCMP will decompress the raw image and output it into one of four formats:

1) decompressed PDS labeled file which contains PDS labels, the histogram object, and an image object, either the browse image or the full image
2) decompressed image file, no labels
3) a decompressed image in the GIF format
4) a decompressed image in the TIFF format

The source code is provided in the SRC subdirectory, of each platform subdirectory. Instructions on how to install and run the software is in the file CLEMDCMP.TXT in the DOC subdirectory, of each platform subdirectory.

Because the image decompression program, CLEMDCMP, requires a Discrete Cosine Transform (DCT) it may take several minutes to decompress an image on hardware platforms with slow processors. For example, in tests on a Macintosh IIci, the decompression takes approximately 4 minutes. CLEMDCMP has been tested on hardware platforms with processors, such as an Intel 486DX2/66-Mhz, and the decompression takes just several seconds.

2.2. Display Software

CLIMDISP in the PCDOS/BIN subdirectory is an image display and processing program. It can be used to display Clementine uncompressed images and histograms. See CLIMDISP.TXT in the PCDOS/DOC subdirectory for instructions on how to install and run the program.

Note: CLIMDISP currently can not create GIF formatted files for the Clementine images. Additionally, it can not read the version of GIF files created by the Clementine Decompression (CLEMDCMP) program which is also included on the Clementine EDR Archive CD-ROMs. If you wish to display Clementine images with CLIMDISP, generate a PDS format image file when decompressing with CLEMDCMP.

A special version of NIH Image, found in the MACSYS7/BIN subdirectory, will display PDS decompressed Clementine images. This program is stored in a Stuffit file which is in BinHex format. See IMAGE.TXT in the DOC subdirectory for instructions on how to install and run the program.

The Clementine EDR image files use the PDS label constructs RECORD_TYPE = "UNK", and ^IMAGE = xxxxx <BYTES> to define the structure of the file. This form of the labels is not supported by the current versions of IMDISP and IMAGE4PDS that are widely distributed by the PDS. To read Clementine decompressed formatted files use the version of IMAGE and CLIMDISP programs that are supplied on this CD-ROM. The Clementine versions CLIMDISP and IMAGE have been tested only on the Clementine data products. No attempt has been made to determine if the Clementine program versions will work on any other PDS data product.

XV is a shareware program for displaying images. XV was written by John Bradley of the University of Pennsylvania. It is in a compressed tar file in the SUNOS/SRC subdirectory. See XV.TXT in the SUNOS/DOC subdirectory for instructions on how to decompress and untar this file. XV will not display PDS labeled files, but will display TIF and GIF formatted files.

The XV software, for image display on a sun/unix environment, is not able to read the Clementine PDS labeled files. If you intend to use XV as the display system for the CLementine data products, output GIF or TIFF images with the CLEMDCMP program.

2.3. SPICE Software

Included on one of the ancillary disks associated with this volume set is the Navigation and Ancillary Information Facility (NAIF) Toolkit and some additional NAIF software. The major component of the NAIF Toolkit is the SPICE Library (SPICELIB), a collection of portable ANSI FORTRAN 77 subroutines. Some of these subroutines are used to read the SPICE kernel files containing Clementine ancillary data, such as spacecraft position, spacecraft attitude, instrument orientation and target body size, shape and orientation. Other SPICELIB subroutines may be used to compute typical observation geometry parameters--such as range, lighting angles, and LAT/LON of camera optic axis intercept on the target body. Several utility programs and SPICELIB demonstration programs are also included in the Toolkit. Versions of this software tested on many popular platforms are provided, as are instructions for porting the code to additional platforms. The FORTRAN subroutines can be called from a user's own application program, whether written in FORTRAN or C, or possibly yet another language. Consult your compiler's Reference Manual for instructions. One of the NAIF programs included in this software collection is PICGEO (for Picture Geometry). It was used to compute all of the geometric parameters appearing in the image labels and index tables. It is included so that users may clearly see the algorithms used in computing these quantities, and so that recalculation of image label geometry parameters using revised algorithms, or adding additional parameters, can be easily achieved.

2.4. Miscellaneous Image Processing Software

MSHELL is an interactive command line and menu driven Image and Signal processing language, developed by ACT Corp., which runs under the Microsoft Windows 3.x or Microsoft NT. MSHELL provides powerful scientific image and signal visualization and processing. A number of custom features were added to the MSHELL Image/Signal Processing Environment to support the Clementine Program. This software is included on one of the ancillary disks associated with this volume set, and will be under a subdirectory of the PCDOS directory.

3. Software Directory Hierarchy

The SOFTWARE subdirectories are based on hardware platforms. Under each platform subdirectory, the executables are in the BIN subdirectory, the source is in the SRC subdirectory and documentation on each program is in the DOC subdirectory. Each DOC subdirectory contains a file, SWINV.CAT which is part of the PDS Software Inventory describing software available within the Planetary Science Community. The contents of the SOFTWARE directory are shown below.

```
[SOFTWARE]
|
|-SOFTINFO.TXT
|
|-[PCDOS]
| |
| |-[BIN]
| |   |
| |   |-CLEMDCMP.EXE
| |   |-CLIMDISP.EXE
| |   |-CLIMDISP.HLP
| |
| |-[SRC]
| |   |
| |   |-CLEMDCMP.C
| |   |-PDS.C
| |   |-BITSTRM.C
| |   |-DECOMP.C
| |   |-HUFFMAN.C
| |   |-WRITEGIF.C
| |   |-PDS.H
| |   |-JPEG_C.H
| |   |-CLEMDCMP.MAK
| |
| |-[DOC]
|     |
|     |-CLEMDCMP.TXT
|     |-CLIMDISP.TXT
|     |-SWINV.CAT
|
|-[MACSYS7]
| |
| |-[BIN]
| | |
| | |-CLEMDEXE.HQX
| | |-IMAGE.HQX
| |
| |-[SRC]
|  |  |
|  |  |-CLEMDSRC.HQX
|  |
|  |-[DOC]
|     |
|     |-CLEMDCMP.TXT
|     |-IMAGE.TXT
|     |-SWINV.CAT
|
|-[SUNOS]
   |
   |-[BIN]
   |  |
   |  |-CLEMDEXE.TZU
   |
   |-[SRC]
   |  |
   |  |-CLEMDSRC.TZU
   |  |-XV3A.TZ
   |
   |-[DOC]
      |
      |-CLEMDCMP.TXT
      |-XV.TXT
      |-SWINV.CAT
```

APPENDIX E

NAIF TOOLKIT DIRECTORY STRUCTURE

This appendix contains the software directory structure of the NAIF Toolkit for a SUN. It is an example of a platform-base model for a single platform. Note that the directory organization shown here does not strictly conform to the recommendations discussed in the *Volume Organization and Naming* chapter of this document.

NAIF

The NAIF directory contains one subdirectory, TOOLKIT. The TOOLKIT tree contains all of the files that make up the NAIF Toolkit.

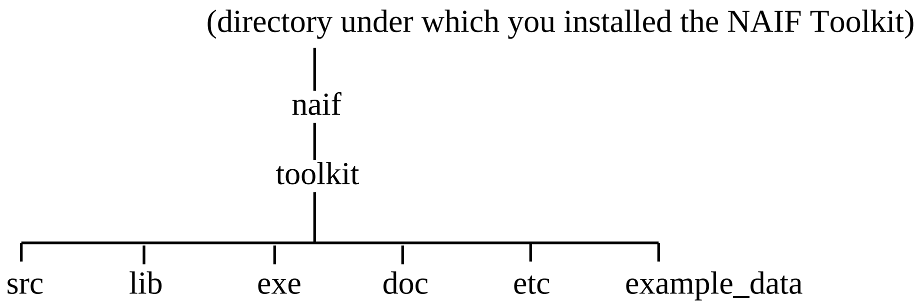
```
(directory under which you installed the NAIF Toolkit)
                          |
                         naif
                          |
                        toolkit
```

TOOLKIT

The TOOLKIT directory contains the file make_toolkit.csh. This is a C shell script that builds all of the object libraries and executables in the TOOLKIT.

```
(directory under which you installed the NAIF Toolkit)
                          |
                         naif
                          |
                        toolkit
                          |
                    make_toolkit.csh
```

TOOLKIT also contains several subdirectories that will be described in more detail in the following sections.



1. SRC
The subdirectories of this directory contain all of the source code for the products in the TOOLKIT.

2. LIB
This directory contains all of the TOOLKIT object libraries.

3. EXE
This directory contains all of the TOOLKIT executables, and where applicable, scripts to run the executables.

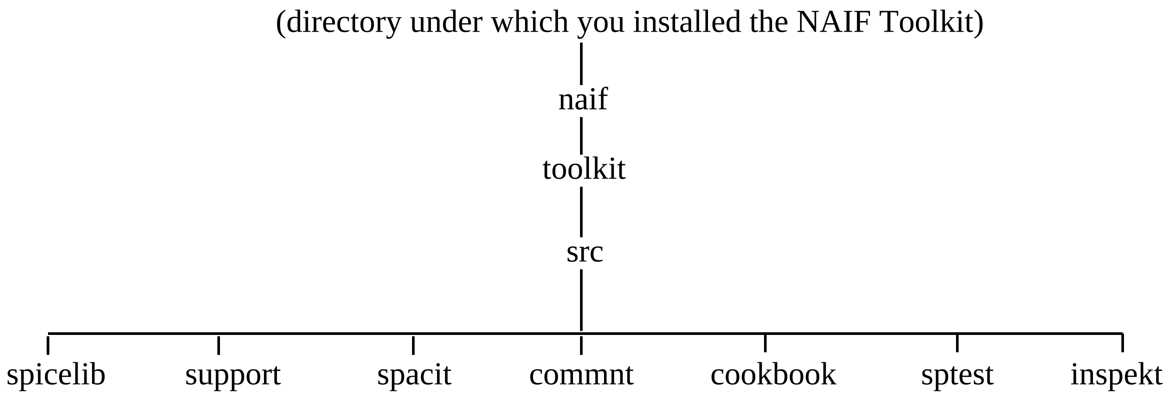
4. DOC
This directory contains all of the TOOLKIT documentation. This includes User's Guides for the programs, Required Reading files for SPICELIB, documents describing the contents of SPICELIB such as the Permuted Index and Module Summary, and documents describing the contents and installation of the Toolkit.

5. ETC
The subdirectories of this directory contain product-specific files that are neither source, documentation, nor data. This includes configuration files, set up files, and help files. The subdirectory build contains the C shell script that creates the toolkit object libraries and executables.

6. EXAMPLE_DATA
This directory contains example data for use with the COOKBOOK and SPTEST programs. These files are to be used only with these programs.

SRC

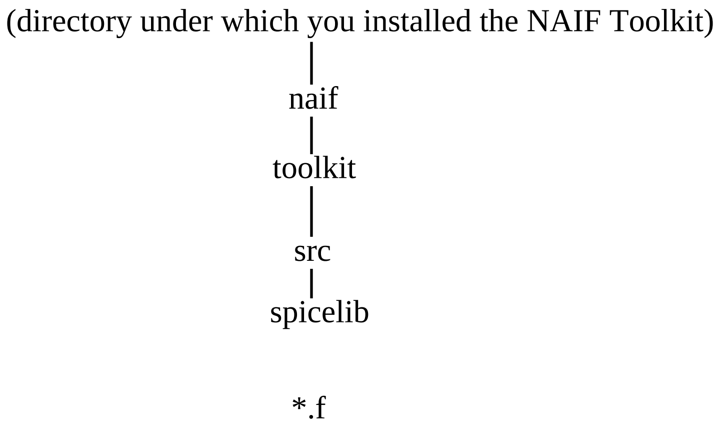
The SRC directory contains one subdirectory for each product in the NAIF Toolkit. Each of these product directories contains the source code files and procedures to create the executable or object library.



SPICELIB

SPICELIB is a Fortran source code library that contains approximately 650 functions, subroutines, and entry points.

This directory contains the SPICELIB source files.



SUPPORT

SUPPORT is a Fortran source code library that contains routines that support the Toolkit programs. These routines are not intended to be used by anyone except NAIF. These routines are not officially supported and may undergo radical changes such as calling sequence changes. They may even be deleted. Do not use them!

This directory contains the SUPPORT library source files.

```
(directory under which you installed the NAIF Toolkit)
                         |
                        naif
                         |
                       toolkit
                         |
                        src
                         |
                      support

                        *.f
```

SPACIT

SPACIT is a utility program that performs three functions: it converts transfer format SPK, CK and EK files to binary format, it converts binary SPK, CK and EK files to transfer format, and it summarizes the contents of binary SPK, CK and EK files.

This directory contains the source code for the SPACIT main program and supporting routines.

```
(directory under which you installed the NAIF Toolkit)
                         |
                        naif
                         |
                       toolkit
                         |
                        src
                         |
                      spaclit

                    spaclit.main

                        *.f
```

COMMNT

COMMNT is a utility program that is used to add comments, extract comments, read comments, or delete comments in SPICE SPK, CK and EK files.

This directory contains the COMMNT main program source file.

```
(directory under which you installed the NAIF Toolkit)
                         |
                        naif
                         |
                       toolkit
                         |
                        src
                         |
                       commnt

                     commnt.main
```

COOKBOOK

The cookbook programs are sample programs that demonstrate how to use SPICELIB routines to obtain state vectors, convert between different time representations, manipulate the comments in binary SPK and CK files, and solve simple geometry problems.

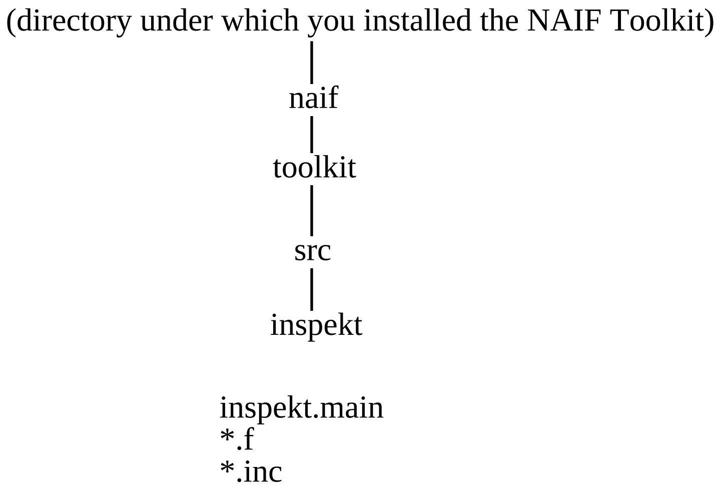
This directory contains the COOKBOOK program source files.

```
(directory under which you installed the NAIF Toolkit)
                         |
                        naif
                         |
                       toolkit
                         |
                        src
                         |
                      cookbook

                     fstspk.main
                     simple.main
                     states.main
                     subpt.main
                     tictoc.main
```

INSPEKT

INSPEKT is a program that allows you to examine the contents of an events component of an E-kernel.

This directory contains the source code for the INSPEKT main program and supporting routines.



(directory under which you installed the NAIF Toolkit)

naif

toolkit

src

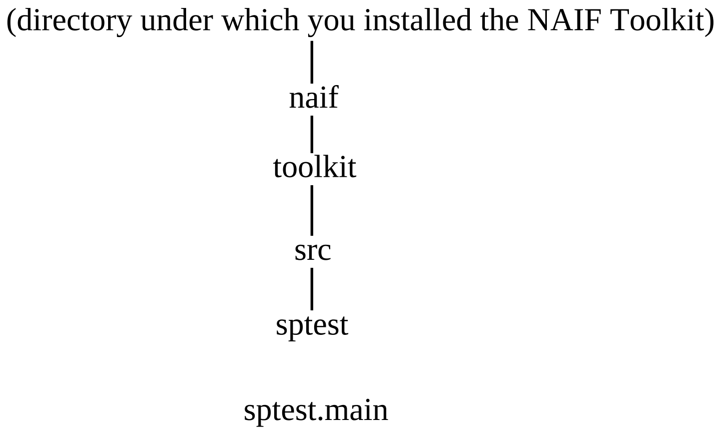
inspekt

inspekt.main
*.f
*.inc

SPTEST

SPTEST is a utility program that tests the SPK file readers by comparing states read on the NAIF VAX with states read on the target machine.

This directory contains the SPTEST program source file.



(directory under which you installed the NAIF Toolkit)

naif

toolkit

src

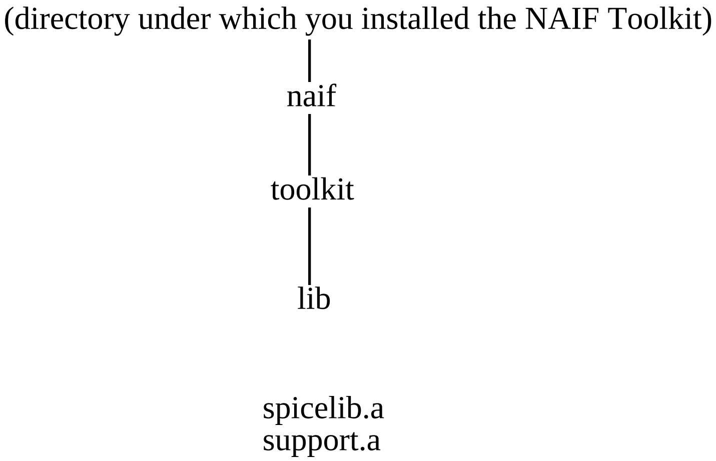
sptest

sptest.main

LIB

---

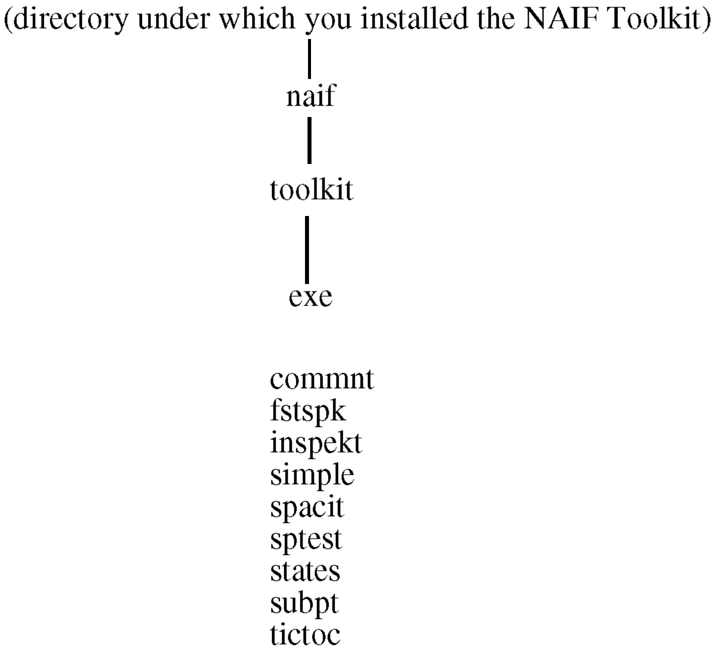
The LIB directory contains spicelib.a, the object library for SPICELIB. It also contains the object library support.a, but this library is for use by the Toolkit programs only. Do not link your applications with it!



EXE

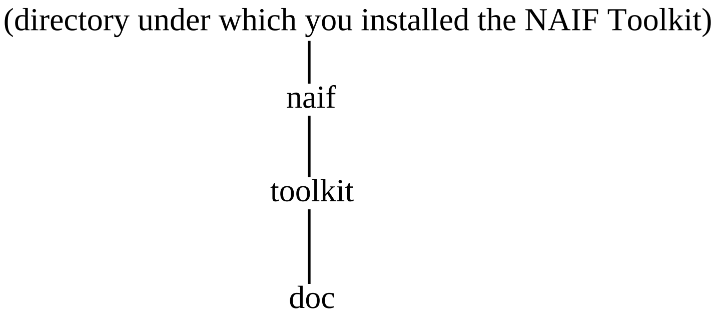
---

The EXE directory contains the NAIF Toolkit executables and, where applicable, scripts to run executables.



DOC

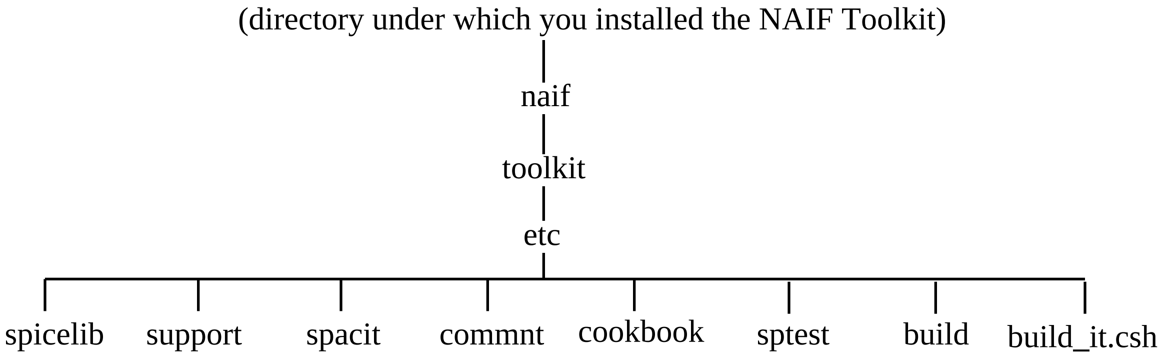
The DOC directory contains all of the TOOLKIT documentation that is available on-line. This includes the user's guides for the programs, all Required Reading files for SPICELIB, all documents describing the contents and porting of SPICELIB, and documents describing the installation and contents of the Toolkit. Please note that the INSPEKT User's Guide is not available on-line.



commnt.ug
fstspk.ug
simple.ug
spacit.ug
sptest.ug
states.ug
subpt.ug
tictoc.ug
*.req
category.txt
libsum.txt
permuted_index.txt
porting.txt
toolkit_install.txt
toolkit_description.txt

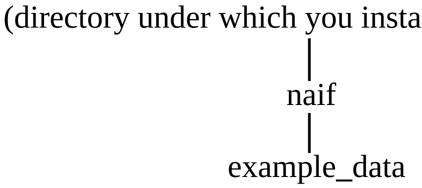
ETC

The ETC directory contains all files for the Toolkit products that are not source, documentation, or data such as set up files, configuration files or help files. It also contains the C shell script used to build the toolkit object libraries and executables.



EXAMPLE_DATA

The EXAMPLE_DATA directory contains all of the NAIF Toolkit data. This data are intended only to be used with the TOOLKIT programs, and are included only to help you get started using the Toolkit.



cook_01.tc
cook_01.tls
cook_01.tpc
cook_01.tsc
cook_01.tsp
cook_02.tc
cook_02.tsp
sptest.gen
sptest.rqs
sptest.tsp

Using the NAIF Toolkit
===

After the installation has been completed successfully, there are a few things that you need to do to get started using SPICELIB. We recommend that you print out the source code for the cookbook programs (./naif/toolkit/src/cookbook/*.main) and examine it. Try running some of the cookbook programs yourself. The cookbook programs demonstrate how to use SPICELIB routines to obtain state vectors, convert between different time representations, manipulate the comments in binary SPK and CK files, and solve simple geometry problems.

Once you're ready to get your hands dirty, you should read the required reading files for SPICELIB. The required reading files are located in the directory ./naif/toolkit/doc and have the extension ``.req''. They are text files that describe families of subroutines and how they interact with the rest of SPICELIB.

The most important required reading files are: TIME, KERNEL, SPK, CK, SCLK, SPC, and NAIF_IDS. You should read at least these.

After you've done these things, you're ready to start programming with SPICELIB!

Appendix -- NAIF's File Naming Conventions
===

NAIF follows a set of conventions for naming files based on the contents of the files. This allows you to find certain types of files in a directory tree quickly.

1. *.for, *.f
Fortran-77 source code files.

2. *.main
Source code files for program modules.

3. *.inc
Fortran-77 include files.

4. *.c
C source code files.

5. *.o
Unix object files.

6. *.obj
VAX/VMS object files.

7. *.a
Unix object library files.

8. *.olb

VAX/VMS object library files.

9. *.tsp
Transfer format SPK (ephemeris) files.

10. *.bsp
Binary format SPK (ephemeris) files.

11. *.tc
Transfer format CK (pointing) files.

12. *.bc
Binary format CK (pointing) files.

13. *.ti
Text IK (instrument parameters) files.

14. *.tls
Leapseconds kernel files.

15. *.tpc
Physical and cartographic constants kernel files.

16. *.tsc
Spacecraft clock coefficients kernel files.

17. *.txt
Text format documentation files.

18. *.ug
Text format User's Guides.

19. *.req
Text format SPICELIB Required Reading files.

20. make_toolkit.csh, build_it.csh
Unix C shell script files for creating the toolkit object libraries and executables.

21. make_toolkit.sh, build_it.sh
Unix Bourne shell script files for creating the toolkit object libraries and executables.

22. (product name)
Unix executable files. For example, spacit is the executable file for the product spacit.

23. make_(product name).com

VAX/VMS command procedures for creating products. For example, make_spicelib.com creates the object library spicelib.olb, while make_spacit.com creates the executable spacit.exe.

24. (product name).exe
VAX/VMS executable files. For example, spacit.exe is the executable file for the product spacit.

These conventions are preliminary. As coordination with AMMOS and the Planetary Data System (PDS) occurs, these conventions may be revised.

APPENDIX F

Acronyms and Abbreviations

The following list contains the acronyms and abbreviations which shall be used in all PDS documentation.

AMMOS	Advanced Multi-Mission Operations System
CCSDS	Consultative Committee for Space Data Systems
CD-ROM	Compact Disc - Read Only Memory
CD-WO	Compact Disk Write Once
CN	Central Node
CODMAC	Committee on Data Management and Computation
DA	Data Administrator
DBA	Database Administrator
DE	Data Engineer
DN	Discipline Node
ECR	Engineering Change Request
GSFC	Goddard Space Flight Center
IDS	Inter-Disciplinary Scientist
ISO	International Standards Organization
JPL	Jet Propulsion Laboratory
NAIF	Navigation and Ancillary Information Facility
NASA	National Aeronautics and Space Administration
NBS	National Bureau of Standards
NSI/DECNET	DEC Network
NSSDC	National Space Science Data Center
ODL	Object Description Language
PC	Personal Computer
PDS	Planetary Data System
PSDD	Planetary Science Data Dictionary

PI	Principal Investigator
PVL	Parameter Value Language
RPIF	Regional Planetary Image Facility
SFDU	Standard Formatted Data Unit
SIS	System Interface Specification
SPICE	Spacecraft, Planetary & Probe Ephemeris, Instrument, C-Matrix, Event File - A system for storing and accessing ancillary information.
SQL	Structured Query Language
UTC	Universal Time Coordinated (often called GMT)
VAX	Virtual Address/Access Extension (DEC Computer)
WORM	Write Once Read Many

Appendix G

SAVED Data

In rare cases data will be encountered in the PDS archive which are classified as having ARCHIVE_STATUS = SAVED. These data are being preserved in a primitive form either pending the production of an archive-quality product, or as part of a save-the-bits campaign for a defunct mission or project. In these cases the most available information has been preserved in as close an approximation of PDS archive format as possible. Following is a description of the criteria applied to datasets considered for safing, and the PDS procedures applied during the process. It is provided here for information only - saved datasets are not considered acceptable for the purposes of meeting PDS archiving requirements.

G.1 Safekeeping Process and Procedures

The decision to save a dataset will normally be made within a discipline node after discussion with the provider of the data. The decision may also be made by a data engineer at the Central Node after discussion with both a data provider and the most relevant discipline node(s). Preservation should take place according to the following procedures.

1. The details of every dataset to be saved will be discussed in a conference, such as a telecon or iteration of e-mail messages, among the data provider, representatives of the relevant discipline node(s), and a data engineer from Central Node. This discussion will address:

    a. The characteristics of the data to be preserved
    b. The reasons for preserving rather than archiving the data
    c. The timetable for producing an archival product from the preserved data
    d. The proposed unique VOLUME_ID for the product
    e. The extent of the additional information to be included.

2. The conclusions of the decision-making conference will be summarized by the data engineer and distributed to all participants.

3. The dataset will normally be prepared and delivered by the data provider according to the agreed content and format.

4. The data engineer at the Central Node will ensure that the product is incorporated into the Distributed Inventory System (DIS).

G.2 Safekeeping Standards

The following items are desirable for any preserved dataset. Some are required.

1. VOLUME_ID - This is required for every preserved product, must be unique within PDS, and must conform to the volume naming standards of PDS.

2. DIS.LBL - This is required for every preserved product and must conform to the PDS labelling standards.

3. AAREADME.TXT - This describes the directory structure of the volume and the content of the volume. It also includes contact information for the original source of the data.

4. INDEX.TXT - This is used if the individual files of data do not have PDS labels. It consists of free format text and is a less rigorous version of INDEX.TAB.

5. Minimal labels - Individual files should be labelled with "minimal labels" as described in Section 5.2.3.

6. Document directory - This is optional but all files must have minimal labels.

7. Software directory - This is optional but all files must have minimal labels

Items 1 through 5 are all strongly recommended for any preserved dataset. Items 6 and 7 are strongly recommended if appropriate. Items 1 and 2 are absolutely required for all preserved datasets.